Handbook of BigDataBench (Version 3.1)—A Big Data Benchmark Suite

Chunjie Luo[1], Wanling Gao[1], Zhen Jia[1], Rui Han[1], Jingwei Li[1], Xinlong Lin[1], Lei Wang[1], Yuqing Zhu[1], and Jianfeng Zhan[1]

[1]Institute of Computing Technology, Chinese Academy of Sciences, China

{ luochunjie, gaowanling, jiazhen, hanrui, lijingwei, linxinlong, wanglei_2011, zhuyuqing, zhanjianfeng}@ict.ac.cn

Abstract. This document presents the handbook of BigDataBench (Version 3.1). BigDataBench is an open-source big data benchmark suite, publicly available from `http://prof.ict.ac.cn/BigDataBench`. After identifying diverse data models and representative big data workloads, BigDataBench proposes several benchmarks specifications to model five important application domains, including search engine, social networks, e-commerce, multimedia data analytics and bioinformatics. BigDataBench (partially) implements the same benchmarks specifications using variety of competitive techniques. The current version BigDataBench 3.1 includes 14 real data sets and the corresponding scalable big data generation tools, and 33 big data workloads. To allow flexible setting and replaying of mixed workloads, BigDataBench provides the multi-tenancy version; To save the benchmarking cost, BigDataBench reduces the full workloads to a subset according to workload characteristics from a specific perspective. It also provides both MARSSx86 and Simics simulator versions for architecture communities.

Keywords: Big Data, Benchmarks, Scale-out workloads, Search Engine, Social Network, E-commerce, Multimedia Data Analytics, Bioinformatics, MapReduce, Spark, MPI, Multi-tenancy, Subsetting, Simulator

1 Introduction

As a multi-discipline research and engineering effort, i.e., system, architecture, and data management, from both industry and academia, BigDataBench is an open-source big data benchmark suite, publicly available from `http://prof.ict.ac.cn/BigDataBench`. In nature, BigDataBench is a benchmark suite for scale-out workloads, different from SPEC CPU (sequential workloads) [17], and PARSEC (multithreaded workloads) [21]. Currently, it simulates five typical and important big data applications: search engine, social network, e-commerce, multimedia data analytics, and bioinformatics. In specifying representative big data workloads, BigDataBench focuses on units of computation that are frequently appearing OLTP, Cloud OLTP, OLAP, interactive and offline analytics in each

application domain. Meanwhile, it takes variety of data models into consideration,, which are extracted from real-world data sets, including unstructured, semi-structured, and structured data. BigDataBench also provides an end-to-end application benchmarking framework [67] to allow the creation of flexible benchmarking scenarios by abstracting data operations and workload patterns, which can be extended to other application domains, e.g., the water consumption application domain in [9].

For the same big data benchmark specifications, different implementations are provided. For example, we and other developers implemented the offline analytics workloads using MapReduce, MPI, Spark, DataMPI, interactive analytics and OLAP workloads using Shark, Impala, and Hive. In addition to real-world data sets, BigDataBench also provides several parallel big data generation tools—BDGS—to generate scalable big data, e.g., a PB scale, from small or medium-scale real-world data while preserving their original characteristics. The current BigDataBench version is 3.1. In total, it involves 14 real-world data sets, and 33 big data workloads.

To model and reproduce the multi-applications or multi-user scenarios on Cloud or datacenters, we provide a multi-tenancy version of BigDataBench, which allows flexible setting and replaying of mixed workloads according to the real workload traces—Facebook, Google and SoGou traces. For system and architecture researches, i. e., architecture, OS, networking and storage, the number of benchmarks will be multiplied according to different implementations, and hence become massive. To reduce the research or benchmarking cost, we select a small number of representative benchmarks, which we call the BigDataBench subset, from a large amount of BigDataBench workloads according to workload characteristics from a specific perspective. For example, for architecture communities, as simulation-based research is very time-consuming, we select a handful number of benchmarks from BigDataBench according to comprehensive micro-architectural characteristics, and provide both MARSSx86 [12] and Simics [15] simulator versions of BigDataBench.

2 Summary of BigDataBench 3.1

BigDataBench is in fast expansion and evolution. Currently, we propose serval benchmark specifications to model five typical application domains, the are available in Section 4. This section summarizes the implemented workloads, their data sets, and scalable data generation tools. The current version BigDataBench 3.1 includes 14 real-world data sets and 33 big data workloads. Table 1 summarizes the real-world and synthetic data sets and scalable data generation tools are included in BigDataBench 3.1, covering the whole spectrum of data types, including structured, semi-structured, and unstructured data, and different data sources, such as text, graph, image, audio, video and table data.

Table 2 presents BigDataBench from perspectives of application domains, operations/ algorithms, data set, software stacks and application types. For some end users, they may just pay attention to specified types of big data applications.

For example, they want to perform an apples-to- apples comparison of software stacks for Offline Analytics. They only need to choose benchmarks with Offline Analytics. On the other hand, if the users want to measure or compare big data systems and architecture, we suggest they cover all benchmarks.

Table 1. The summary of data sets and data generation tools.

No.	data sets	data set description[1]	scalable data set
1	Wikipedia Entries [18]	4,300,000 English articles (unstructured text)	Text Generator of BDGS
2	Amazon Movie Reviews [8]	7,911,684 reviews (semi-structured text)	Text Generator of BDGS
3	Google Web Graph [11]	875713 nodes, 5105039 edges (unstructured graph)	Graph Generator of BDGS
4	Facebook Social Network [10]	4039 nodes, 88234 edges (unstructured graph)	Graph Generator of BDGS
5	E-commerce Transaction Data	Table 1: 4 columns, 38658 rows. Table 2: 6 columns, 242735 rows (structured table)	Table Generator of BDGS
6	ProfSearch Person Resumés	278956 resumés (semi-structured table)	Table Generator of BDGS
7	ImageNet [25]	ILSVRC2014 DET image dataset (unstructured image)	ongoing development
8	English broadcasting audio files [1]	Sampled at 16 kHz, 16-bit linear sampling (unstructured audio)	ongoing development
9	DVD Input Streams [2]	110 input streams, resolution:704*480 (unstructured video)	ongoing development
10	Image scene [3]	39 image scene description files (unstructured text)	ongoing development
11	Genome sequence data [4]	cfa data format (unstructured text)	4 volumes of data sets
12	Assembly of the human genome[5]	fa data format (unstructured text)	4 volumes of data sets
13	SoGou Data [16]	the corpus and search query data from SoGou Labs (unstructured text)	ongoing development
14	MNIST [13]	handwritten digits database which has 60,000 training examples and 10,000 test examples (unstructured image)	ongoing development

[1]The further detail of data schema is available from Section 5

Table 2. The summary of the implemented workloads in BigDataBench 3.1.

Domains	Operations or Algorithm	Types	Data Set	Software Stacks	ID[1]
Search Engine	Grep	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-1
	WordCount	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-2
	Index	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-4
	PageRank	Offline Analytics	Google Web Graph	MPI, Spark, Hadoop	W1-5
	Nutch Server	Online Service	SoGou Data	Nutch	W1-6
	Sort	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-7
	Read	Cloud OLTP	ProfSearch Resumes	HBase, Mysql	W1-11-1
	Write	Cloud OLTP	ProfSearch Resumes	HBase, Mysql	W1-11-2
	Scan	Cloud OLTP	ProfSearch Resumes	HBase, Mysql	W1-11-3
Social Network	CC	Offline Analytics	Facebook Social Network	MPI, Spark, Hadoop	W2-8-1
	Kmeans	Offline Analytics	Facebook Social Network	MPI, Spark, Hadoop	W2-8-2
	BFS	Offline Analytics	Self Generating by the program	MPI	W2-9
E-commerce	Select Query	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-1
	Aggregation Query	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-2
	Join Query	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-3
	CF	Offline Analytics	Amazon Movie Review	hadoop, Spark, MPI	W3-4
	Bayes	Offline Analytics	Amazon Movie Review	hadoop, Spark, MPI	W3-5
	Project	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-1
	Filter	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-2
	Cross Product	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-3
	OrderBy	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-4
	Union	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-5
	Difference	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-6
	Aggregation	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-7
Multimedia	BasicMPEG	Offline Analytics	stream data	Libc	W4-1
	SIFT	Offline Analytics	ImageNet	MPI	W4-2-1
	DBN	Offline Analytics	MNIST	MPI	W4-2-2
	Speech Recognition	Offline Analytics	audio files	MPI	W4-3
	Ray Tracing	Offline Analytics	scene description files	MPI	W4-4
	Image Segmentation	Offline Analytics	ImageNet	MPI	W4-5
	Face Detection	Offline Analytics	ImageNet	MPI	W4-6
Bio-informatics	SAND	Offline Analytics	Genome sequence data	Work Queue	W5-1
	BLAST	Offline Analytics	Assembly of the human genome	MPI	W5-2

[1]The workload ID of BigDataBench 3.1 corresponds with the workload ID in the BigDataBench specification which can be found at Section 4

2.1 What are the differences of BigDataBench from other benchmark suites?

As shown on the Table 3, among the ten desired properties, the BigDataBench is more sophisticated than other state of art big data benchmarks.

Table 3. The summary of different Big Data Benchmarks.

	Specification	Application domains	Workload types	Workloads	Scalable data sets abstracting from real data	Multiple implementations	Multi-tenancy	Subsets	Simulator version
BigData Bench	Y	five	four[1]	thirty-three[2]	eight[3]	Y	Y	Y	Y
BigBench	Y	one	three	ten	three	N	N	N	N
CloudSuite	N	N/A	two	eight	three	N	N	N	Y
HiBench	N	N/A	two	ten	three	N	N	N	N
CALDA	Y	N/A	one	five	N/A	Y	N	N	N
YCSB	Y	N/A	one	six	N/A	Y	N	N	N
LinkBench	Y	N/A	one	ten	one	Y	N	N	N
AMP Benchmarks	Y	N/A	one	four	N/A	Y	N	N	N

[1]The four workload types is Offline Analytics, Cloud OLTP, Interactive Analytics and Online Service

[2]There are 42 workloads in the specification. We have implemented 33 workloads

[3]There are 8 real data sets can be scalable, other 6 ones are ongoing development

2.2 BigDataBench Evolution

As shown in Fig. 1, the evolution of BigDataBench has gone through four major stages:

At the first version, we released three benchmarks, BigDataBench 1.0 (6 workloads from Search engine), DCBench 1.0 (11 workloads from data analytics), and CloudRank 1.0 (mixed data analytics workloads).

At the second version, we combined the previous three benchmarks and release BigDataBench 2.0, through investigating the top three important application domains from internet services in terms of the number of page views and daily visitors. BigDataBench 2.0 is a big data benchmark suite from internet services. It includes 6 real-world data sets, and 19 big data workloads with different implementations, covering six application scenarios: micro benchmarks, Cloud OLTP, relational query, search engine, social networks, and e-commerce. Moreover, BigDataBench 2.0 provides several big data generation tools–BDGS– to generate scalable big data, e.g, PB scale, from small-scale real-world data while preserving their original characteristics.

In BigDataBench 3.0, we made a multidisciplinary effort to the third version– BigDataBench 3.0, which includes 6 real-world, 2 synthetic data sets, and 32 big data workloads, covering micro and application benchmarks from typical application domains, e. g., search engine, social networks, and e-commerce. As to generating representative and variety of big data workloads, BigDataBench 3.0 focuses on units of computation that frequently appear in Cloud OLTP, OLAP, interactive and offline analytics.

Now, we release the fourth version, BigDataBench 3.1. It includes 5 application domains, not only the three most important application domains from internet services, but also emerging and important domains (Multimedia analytics and Bioinformatics), altogether 14 data sets and 33 workloads. The Multi tenancy version for Cloud computing communities and simulator version for architecture communities are also released.

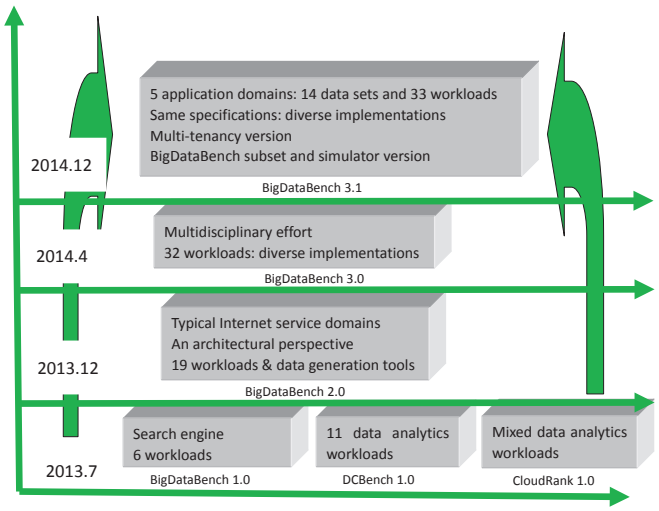


Fig. 1. BigDataBench Evolution

2.3 What is new in BigDataBench 3.1

We updated the Benchmarking methodology and added two new application domains: Multimedia and Bioinformatics. Now, there are five typical application domains: Search Engine, Social Network, E-commerce, Multimedia and Bioinformatics in BigDataBench 3.1. With the new methodology, we proposed the Benchmark specification for each application domain, and defined data sets and workloads in the application domains. Based on the specification, we implemented the BigDataBench 3.1. Now it includes 14 real-world data sets, and 33

big data workloads. The Multi tenancy version for Cloud computing communities and simulator version for architecture communities are also released.

3 Big Data Benchmarking Methodology

Figure 2 summarizes the benchmarking methodology in BigDataBench. Overall, it involves five steps: investigating and choosing important application domains; identifying typical workloads and data sets; proposing big data benchmarks specifications; providing diverse implementations using competitive techniques; mixing different workloads to assemble multi-tenancy workloads or subsetting big data benchmarks.

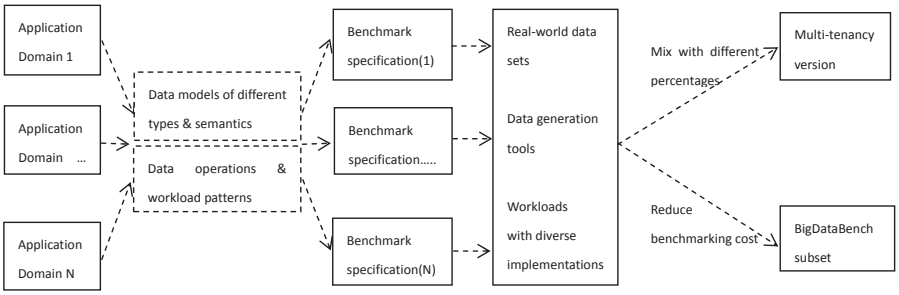


Fig. 2. BigDataBench benchmarking methodology.

At the first step, we investigated typical applications domains. First of all, we investigated the dominant application domains of internet services—an important class of big data applications according to widely acceptable metrics—the number of page views and daily visitors. According to the analysis in [7], the top three application domains are search engines, social networks, and e-commerce, taking up 80% page views of all the internet services in total. Meanwhile, multimedia data analytics and bioinformatics are two emerging but important big data application domains. So we selected out those five important applications domains: search engine, social network, e-commerce, multimedia data analytics and bioinformatics. At the second step, we analyzed typical workloads and data sets in each domain from two perspectives: diverse data models of different types, i.e., structured, semi-structured, and unstructured, and different semantics, e.g., text, graph, multimedia data; identifing frequent-appearing data operations and workload patterns. After that, we proposed big data benchmarks specifications for each domain. At the fourth step, we implemented the same specifications using competitive techniques. For example, for offline analytics workloads, we implemented the workloads using MapReduce, MPI, Spark, DataMPI. Meanwhile, we choosed real-world data sets, and then provides parallel big data generation tools to generate scalable big data while preserving their original characteristics.

Finally, we provided the multi-tenancy version and the BigDatabench subset for different purposes. We provide the multi-tenancy version of BigDataBench, which allows flexible setting and replaying of mixed workloads with different percentages. To reduce the research or benchmarking cost, we select a small number of representative benchmarks, which we call the BigDataBench subset, from a large amount of BigDataBench workloads according to workload characteristics from a specific perspective.

4 BigDataBench specification

There are five application domains in BigDataBench, namely search engine, social network, e-commence, multimedia, and bioinformatics. In this section, we will describe the details of each scene by which the implementation of BigDataBench is guided. When describing the workloads, we use natural language in English.

4.1 Search Engine

Web search engine is used to search information from HTML markup of the web pages which are crawled by spider. As shown in Figure 3, there are two scenarios, general search and vertical search, in BigDataBench. While general search indexes all the web pages of the Internet and returns thousands of links for a query, vertical search indexes content specialized by topic, and delivers more relevant results to the user. To achieve this, vertical search need filter the pages with special topic and extract semantic information.

Figure 4 shows the details of the search engine in BigDataBench. The data which search engine mainly process are web pages. There are three additional data : meta table, index, and search log. The meta table contains the attributes of pages which are derived from the original web pages. The details of meta table are shown in Table 4.

In BigDataBench, web pages are generated by data generator instead of being downloaded from Internet. After obtaining a web page, search engine analyzes each page to obtain the text contents and the structure of the web graph. The text contents are then indexed by the search engine, while the web graph are used to compute the importance of each page. When users send queries to a search engine, the engine examines its index and provides listing pages which are sorted according to the importance of pages and the relevance between queries and the pages. It is not easy for users to give effective queries to search engine. Users need to be familiar with specific terminology in a knowledge domain or try different queries until they are satisfied with the results. To solve the problem, web search engines often recommend search queries [31, 65, 42] to users according to the historical search records. Additionally, a web search engine often returns thousands of pages, which makes it difficult for users to browse or to identify relevant information. Clustering or Classifying methods can automatically group the results into a list of meaningful categories, so that users can filter the results

to a special category they are interested in. This is achieved by Vivisimo, Carrot2 etc. Moreover, vertical search engine, which focuses on a specific segment of online content, are included in BigDataBench. To achieve vertical search, the pages with special topic are selected out, and then semantic information are extracted. The semantic information can then be accessed directly by the users.

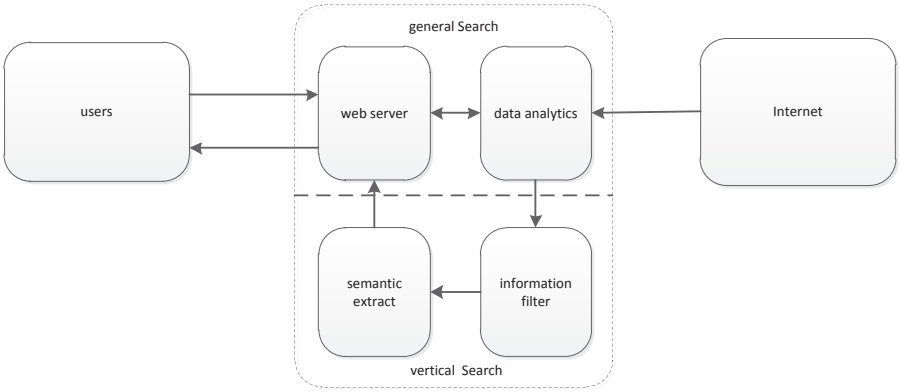


Fig. 3. Abstraction of search engine in BigDataBench.

Table 4. The meta table.

attribute	description
content	the text content of the page without html tags.
URL	the URL of the page.
out_link	the out links of the page.
score	the result of page rank
category	the topic category of the page.
key word	the key words of the page.

Workloads:

W1-1: Parsing. Extract the text contents and out links from the raw web pages. Parsring is the first thing to do after downloading the raw web pages in search engine. This can be done by using regex expression to search some pattern of html tags. This can also be seen as the string search which is widely used in text search engine.

W1-2: Statistic. Count the word frequency to extract the key word which represents the features of the page.

W1-3: Classification. Classify text contents into different categories.

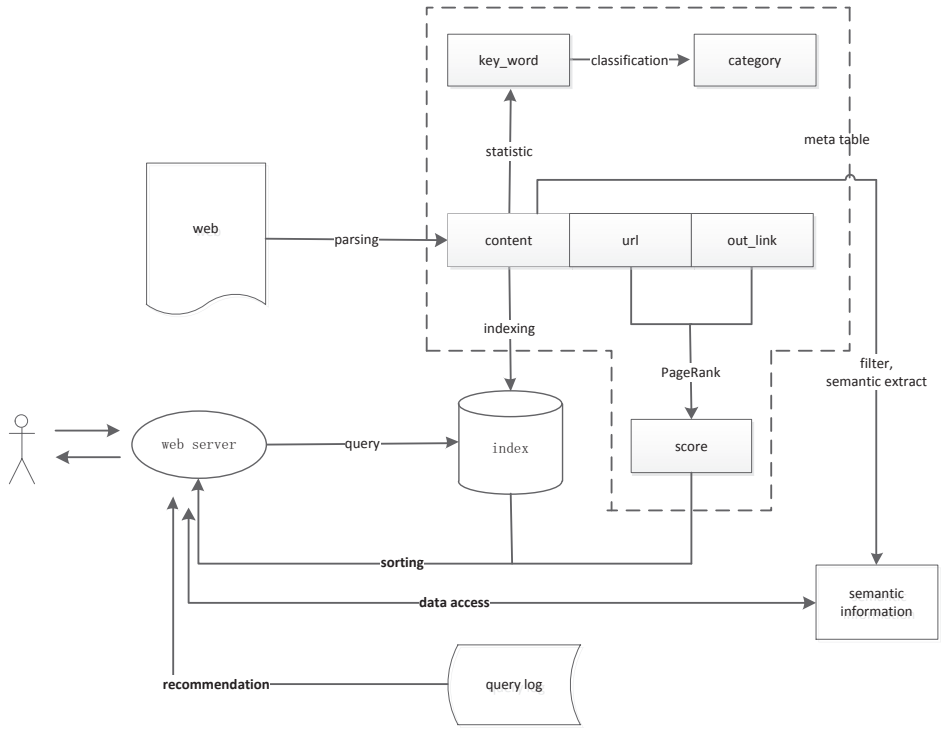


Fig. 4. Process of search engine.

W1-4: Indexing. The process of creating the mapping of term to document id lists.

W1-5: PageRank. Compute the importance of the page according to the web link graph using PageRank. The web graph is built by the out links of each page.

W1-6: Search query. The online web search server.

W1-7: Sorting. Sort the results according to the page ranks and the relevance between queries and documents.

W1-8: Recommendation. Recommend related search queries to users by mining the search log.

W1-9: Filter. Identify pages with specific topic which can be used for vertical search.

W1-10: Semantic extract. Extract semantic information.

W1-11: Data access. Read, write, and scan the semantic information.

4.2 Social Network

Social media allows people to create, share or exchange information, ideas in virtual communities and networks (consumer generated media). We use the

the application of microblogging in our social network domain. Users register by providing some basic information. The users then can follow other users or be followed by other users. In this way, they form lots of communities in virtual. And users can post their tweets to share their information in their communities. The owner of the platform analyze the large network and content of tweets to supply better services, for example, finding communities, recommending friends, classifying the sentiment of a tweet, finding the hot topic, active users and the leaders of opinion. The diagram of social network is shown in Figure 5.

In the social network scene of BigDataBench, there are three tables, the user table, the relation table and the tweet table. The dependence of these table can be seen in Figure 6. And the details are shown in Table

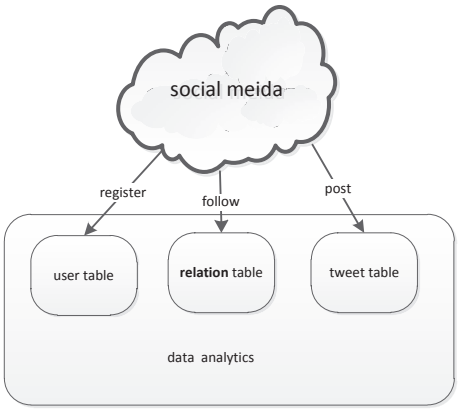


Fig. 5. Abstraction of social network in BigDataBench.

Table 5. The user table.

attribute	description
user_id	the id of the user
sex	the sex of the user
age	the age of the user
education	the situation of education
tag	the terms showing characteristics of the user

workloads

W2-1: Hot review topic. Select the top N tweets by the number of review

W2-2: Hot transmit topic. Select the tweets which are transmitted more than N times.

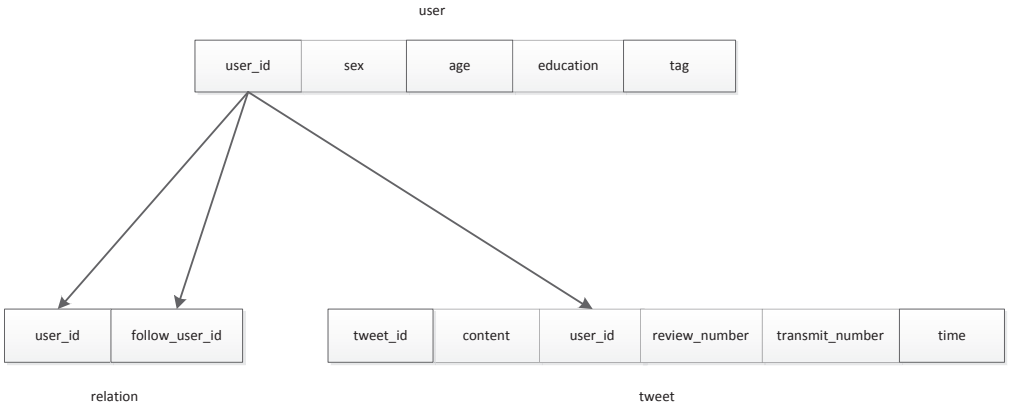


Fig. 6. Tables used in social network scene.

Table 6. The relation table.

attribute	description
user_id	the id of the user
follow_user_id	the user id who is followed

W2-3: Active user. Select the top N person who post the largest number of tweets.

W2-4: Leader of opinion. Select top ones whose number of review and transmit are both large than N.

W2-5: Topic classify. Classify the tweets to certain categories according to the topic.

W2-6: Sentiment classify. Classify the tweets to negative or positive according to the sentiment.

W2-7: Friend recommendation. Recommend friend to person according the relational graph.

W2-8: Community detection . Detecting clusters or communities in large social networks.

Table 7. The tweet table.

attribute	description
tweet_id	the id of the tweet
content	the content of the tweet
user_id	the id of user who own the tweet
review_number	the number of review
transmit_number	the number of transmitting
time	the publish time of the tweet

W2-9: Breadth first search. Sort persons according to the distance between two people.

4.3 E-commence

In the E-commence domain in BigDataBench, as shown in 7, buyers order for goods and review them. A order refers to a action of buying. In a order, there may be lots of items, and each item is related to a specific goods. As in Figure 8, for example, one order may include one pen and two books. In the E-commence, there are many statistic workloads to provide business intelligence. And the review analysis and recommendation are also the typical application of the E-commence. The details of the data are described in Table 8 and Table 9.

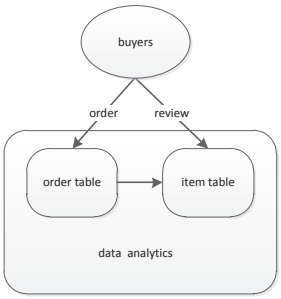


Fig. 7. Abstraction of E-commence in BigDataBench.

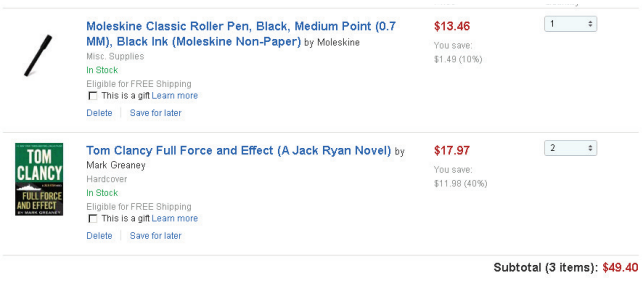

Moleskine Classic Roller Pen, Black, Medium Point (0.7 MM), Black Ink (Moleskine Non-Paper) by Moleskine
Misc. Supplies
In Stock
Eligible for FREE Shipping
This is a gift Learn more
Delete Save for later
$13.46
You save: $1.49 (10%)
1
Tom Clancy Full Force and Effect (A Jack Ryan Novel) by Mark Greaney
Hardcover
In Stock
Eligible for FREE Shipping
This is a gift Learn more
Delete Save for later
$17.97
You save: $11.98 (40%)
2
Subtotal (3 items): $49.40

Fig. 8. The order and item example.

Workloads The workloads are similar as queries used in [55] but are specified in the E-commence environment. Moreover, two workloads, namely recommendation and sensitive classification, are added since they are very popular in the E-commence.

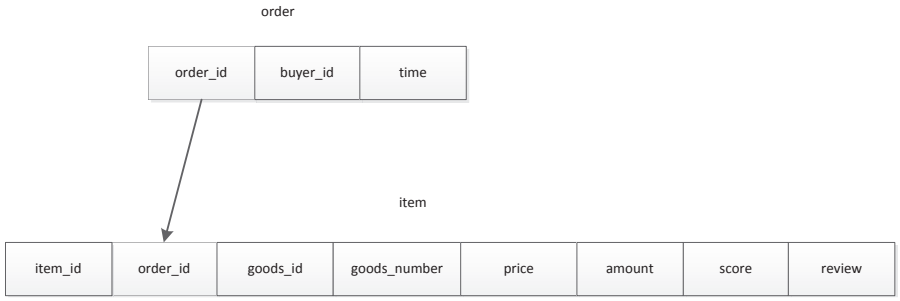


Fig. 9. Tables used in E-commence scene.

Table 8. The order table.

attribute	description
order_id	the id of the order
buyer_id	the id of person who own the order
time	the time of the order occurred

W3-1: Select query. Find the items of which the sales amount is over 100 in a single order.

W3-2: Aggregation query. Count the sales number of each goods.

W3-3: Join query. Count the number of each goods that each buyer purchased between certain period of time.

W3-4: Recommendation. Predict the preferences of the buyer and recommend goods to them .

W3-5: Sensitive classification. Identify positive or negative review.

W3-6: Basic data operation. Unit of operation of the data.

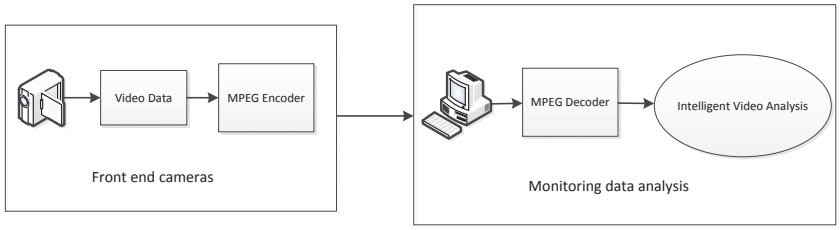
4.4 Multimedia

In the multimedia domain, we simulate an intelligent video surveillance scenario.

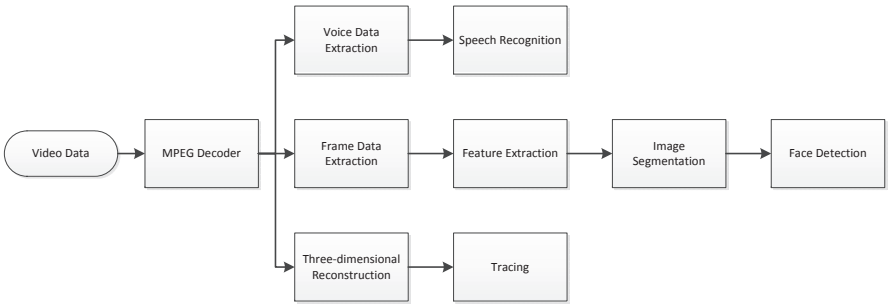
Table 9. The item table.

attribute	description
item_id	the id of the item
order_id	the id of order which the item belongs to
goods_id	the id of goods
goods_number	the number of goods
price	the price of goods
amount	the total assumption of the item
score	the score the buyer gave
review	the text commence the buyer gave

Fig. 10 summarises the brief process of a video surveillance system. The overall framework is shown as Fig. 10(a). First, the gathered video data from front end cameras are delivered to MPEG encoder, and generating MPEG video streams. Then the video streams are packaged for transmission. When the computer receives the video streams, it first decodes the streams using MPEG decoder. Next, a series of analysis can be conducted to dig information. Fig. 10(b) presents the process of intelligent video analysis. Three kinds of analysis can be done to monitor potential anomalies. The first one is to analyze the voice data so as to detect the sensitive words; The second one is three-dimensional reconstruction to get the contour of the monitoring scenario. The third one is to analyze the video frame data and dig the information we concerned about.



(a) Overall Framework



(b) Intelligent Video Analysis.

Fig. 10. Brief process of video surveillance.

Data Model:

Fig. 11 describes the data model of multimedia domain, which is one of the major components. The three cornerstone aspects of multimedia data types are video, audio, and image. The audio data and image data in our domain are derived from the video monitoring data. Video data is an illusion of movement by playing a sequence of frames in quick succession, which are in fact a series of still images. An analog image can be transformed into a digital image after sampling and quantization, which consists of pixels. Audio data also needs

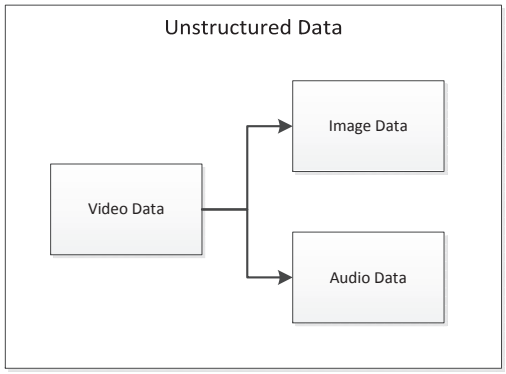


Fig. 11. Data Model of Multimedia.

analog-to-digital convertion.

Workloads:

W4-1: MPEG Decoder. We include a workload undoing the encoding to retrieve original video data. For example, MPEG-2 is a standard for video compression and associated audio data compression.

W4-2: Feature extraction. Workloads for this purpose is mainly extracting the characteristics of video frames and representing original redundant data using features vector.

W4-3: Speech Recognition. This workload targets at content identification of associated audio data, and translating speech into text.

W4-4: Ray Tracing. We include a workload for simulating 3-Dimensional Scene, such as panoramic monitoring using many cameras, which can simulate real scenarios and make for deeper analysis.

W4-5: Image Segmentation. This workload is used to divide the video frames to several regions which can simplify their representation and make them easier to analyze.

W4-6: Face Detection. We include a workload for detecting faces in the video frames.

4.5 Bioinformatics

In the bioinformatics domain, we simulate a genome detection scenario, which is a promising domain of disease prevention and treatment. Fig. 12 describes the brief process of genome detection. We temporarily omit some details, such as specimen collection, DNA extraction and sequence format conversion. There are two important processes of genome detection, gene sequencing and sequence alignment. Gene sequencing is to determine the order of four bases in a strand of DNA. Sequence assembly and mapping are two basic methods of the next-generation sequencing technology. information technology. High-throughput bi-

ological technologies generate an exponentially growing amount of big data. The total amount of DNA sequenced of human, animals and plants exceeds 2000 trillion bases. Analysing and processing these big genome data quickly and accurately is of great significance, for the genome of the organism contains all genetic information about their growth and development. An promising application domain of genome detection is disease prevention and treatment.

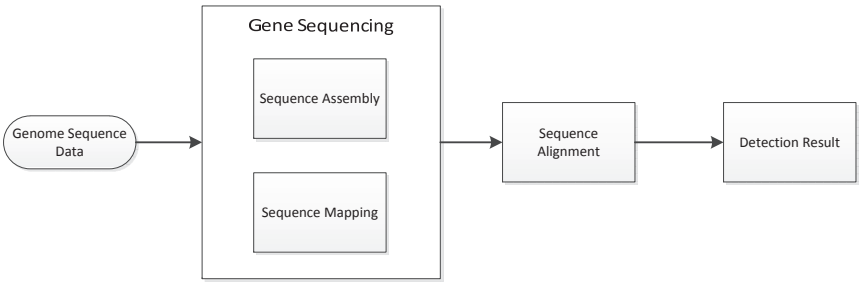


Fig. 12. Brief process of genome detection.

Workloads:

W5-1: Sequence assembly. In this process, We include a workload to align and merge multiple fragments into the original DNA sequence. Generally, a DNA sequence is broken into millions of fragments, and sequence assembly technology is used to recombine these fragments according to contigs.

W5-2: Sequence alignment. This is used to identify the similarity of multiple DNA sequences, and expose the relationship considering of function, structure and evolution. Sequence alignment includes pair-wise comparison and multiple alignments from the perspective of sequence numbers, and partial/global comparison from the perspective of regions.

5 Benchmark implementation

Based on the specification, we implemented the BigDataBench 3.1 workloads. The implementation of the specification is incomplete in current version, and we will use unified data to complete the implementation of the respective workloads of the five domain in BigDataBench.

5.1 Search Engine

As shown in Figure 13, we use the data sets of Wikipedia Entries and Google Web Graph as the input data of the analytics workloads in general search, and use Personal Resumes as the data of vertical search. To generate search query

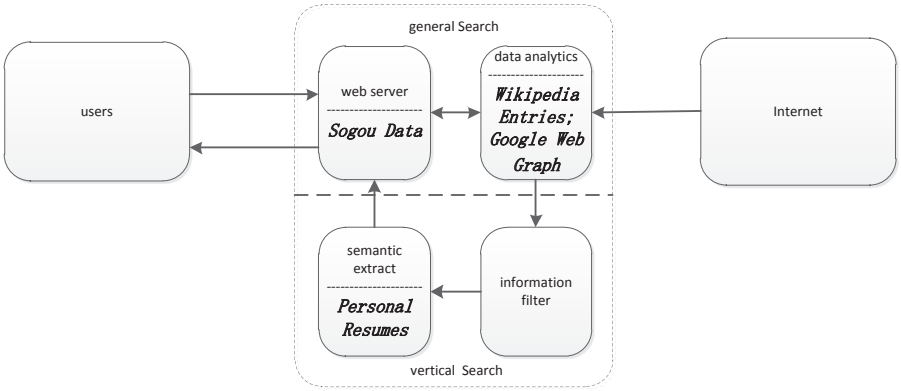


Fig. 13. Real data sets used in BigDataBench.

and provide search services, we use Sogou Data as the original data. The details of these data sets are described as following:

Wikipedia Entries[18]. The Wikipedia data set is unstructured text, with 4,300,000 English articles. The content of Wikipedia articles included: Arts, Biography, Geography, History, Mathematics, Science, Society and Technology.

Google Web Graph [11]. This data set is unstructured, containing 875713 nodes representing web pages and 5105039 edges representing the links between web pages. This data set is released by Google as a part of Google Programming Contest.

Personal Resumes. This data is from a vertical search engine for scientists developed by ourselves. The data set is semi-structured, consisting of 278956 resumes automatically extracted from 20,000,000 web pages of university and research institutions. The resume data have fields of name, email, telephone, address, date of birth, home place, institute, title, research interest, education experience, work experience, and publications.

Sogou Data [16]. This data set is unstructured, including corpus and search query data from Sogou Lab, based on the corpus we gotten the index and segment data which the total data size is 4.98GB.

According with the specification of search engine and data sets, we implement the W1-1, W1-2, W1-4, W1-5, W1-6, W1-7 and W1-11 workloads on various software stack. The details of the implementation of the workloads are shown in Table 10.

Table 10. The summary of search engine workloads.

ID	Implementation	Description	Data set	Software stack
W1-1	Grep	String searching used to parser web pages	Wikipedia data	MPI, Spark, Hadoop
W1-2	WordCount	Counting the word frequency to do statistic	Wikipedia Data	MPI, Spark, Hadoop
W1-4	Index	Indexing web pages for searching	Wikipedia data	MPI, Spark, Hadoop
W1-5	PageRank	Computing the importance of the page	Google Web Graph	MPI, Spark, Hadoop
W1-6	Nutch Server	Providing online search services	Sogou Data	Nutch
W1-7	Sort	Ordering the data	Wikipedia data	MPI, Spark, Hadoop
W1-9-1	Read	Read operation of data access	Personal Resumes	HBase, Mysql
W1-9-2	Write	Write operators of data access	Personal Resumes	HBase, Mysql
W1-9-3	Scan	Scan operators of data access	Personal Resumes	HBase, Mysql

5.2 Social Network

Currently, we only implement the W2-8 and W2-9 workloads and we will complete the implementation soon. We use Facebook Social Network as the input data of workload of W2-8 which is implemented using two different algorithms. And we also use the implementation of breadth first search in Graph500 as the W2-9 workload. **Facebook Social Network** [10] contains 4039 nodes, which represent users, and 88234 edges, which represent friendship between users. The details of the implementation of the workloads are shown in Table 11.

Table 11. The summary of social network workloads.

ID	Implementation	Description	Data set	Software stack
W2-8-1	CC	Community detection using Connect Component algorithm	Facebook Social Network	MPI, Spark, Hadoop
W2-8-2	Kmeans	Community detection using Kmeans algorithm	Facebook Social Network	MPI, Spark, Hadoop
W2-9	BFS	Breadth first search	synthetic graph	MPI

5.3 E-commence

We implement all the workloads according to the specification of E-commence. As shown on the Table 12, we use E-commerce Transaction data. However, there is no reviews in this data. As a result, we use the Amazon Movie Reviews data as the attribute of score and review in the item table to implement the workloads of W3-4 and W3-5. The E-commence Transaction and Amazon Movie Reviews are described as following:

E-commence Transaction. This data set is from an E-commerce web site, which we keep anonymous by request. This data set is structured, consisting of two tables: ORDER and order ITEM. The detail is shown in table 12.

Table 12. Schema of E-commence transaction data

ORDER	ITEM
ORDER_ID INT	ITEM_ID INT
BUYER_ID INT	ORDER_ID INT
CREATE_ID DATE DATE	GOODS_ID INT
	GOODS_NUMBER NUMBER(10,2)
	GOODS_PRICE NUMBER(10,2)
	GOODS_AMOUNT NUMBER(14,6)

Amazon Movie Reviews[8] This data set is semi-structured, consisting of 7,911,684 reviews on 889,176 movies by 253,059 users. The data span from Aug 1997 to Oct 2012. Fig 14 shows the data format. The raw format is text, and consists of productID, userID, profileName, helpfulness, score, time, summary and text.

The details of the implementation of the workloads are shown in Table 13.

```
product/productId: B00006HAXW
review/userId: A1RSDE90N6RSZF
review/profileName: Joseph M. Kotow
review/helpfulness: 9/9
review/score: 5.0
review/time: 1042502400
review/summary: Pittsburgh - Home of the OLDIES
review/text: I have all of the doo wop DVD's and this
one is as good or better than the 1st ones. Remember
once these performers are gone, we'll never get to see
them again.Rhino did an excellent job and if you like or
love doo wop and Rock n Roll you'll LOVE this DVD !!
```

Fig. 14. Excerpt of movie review data set

Table 13. The summary of E-commence workloads.

ID	Implementation	Description	Data set	Software stack
W3-1	Select query	Find the items of which the sales amount is over 100 in a single order	E-commence Transaction	Hive, Shark, Impala
W3-2	Aggregation query	Count the sales number of each goods	E-commence Transaction	Hive, Shark, Impala
W3-3	Join query	Count the number of each goods that each buyer purchased between certain period of time	E-commence Transaction	Hive, Shark, Impala
W3-4	CF	Recommendation using Collaborative Filtering algorithm	Amazon Movie Reviews	Hadoop, Spark, MPI
W3-5	Native Bayes	Sensitive classification using Native Bayes algorithm	Amazon Movie Reviews	Hadoop, Spark, MPI
W3-6-1	Project	Basic operator	E-commerce Transaction	Hive, Shark, Impala
W3-6-2	Filter	Basic operator	E-commerce Transaction	Hive, Shark, Impala
W3-6-3	Cross Product	Basic operator	E-commerce Transaction	Hive, Shark, Impala
W3-6-4	OrderBy	Basic operator	E-commerce Transaction	Hive, Shark, Impala
W3-6-5	Union	Basic operator	E-commerce Transaction	Hive, Shark, Impala
W3-6-6	Difference	Basic operator	E-commerce Transaction	Hive, Shark, Impala
W3-6-7	Aggregation	Basic operator	E-commerce Transaction	Hive, Shark, Impala

5.4 Multimedia

The data processing flow is shown in Fig. 15. The received video streams are decoded to get original video data. Then one branch is to analyze a sequence of video frames which in fact are a series of static images. The second branch is to extract and analyze corresponding audio data. The third branch is to reconstruct three-dimensional scene.

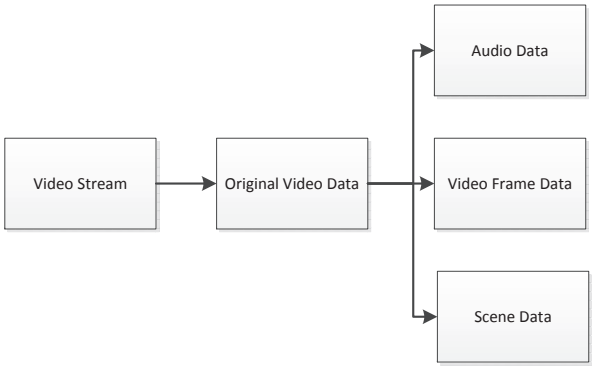


Fig. 15. Data processing flow of video surveillance.

Since we don't own the real surveillance videos, we use similar patterns of data for present. For BasicMPEG, we choose the DVD Input Streams data. For the branch of analyzing video frames, we choose ImageNet [25], which is influential and comprehensive, as the video frame data for Feature extraction, Image Segmentation and Face Detection. The corresponding audio data needs large vocabulary and relatively standard pronunciation to conform to the real scene, in that case, we choose English broadcasting audio data for Speech Recognition. Three-dimensional reconstruction needs scene description files, so we choose the Image scene data for Ray Tracing. Surveillance videos for traffic involve in car license number recognition, then we choose MNIST for DBN. These data sets are described as following:

ImageNet[25].This data set is unstructured, organized according to the WordNet hierarchy, with 21841 non-empty synsets, including categories of plant, formation, natural object, sport, artifact, fun guns, person, animal, and Misc, adding up to 14197122 images.

English broadcasting audio files[1] . This audio data set is unstructured, containing about 8000 audio files from VOA, BBC, CNN, CRI and NPR.

DVD Input Streams[2]. This data set is unstructured, consisting of 110 input streams, with the resolution of 704*480. The contents of the streams include cactus and comb, flowers, mobile and calendar, table tennis, et al.

Image scene[3]. This data set is semi-structured, with 39 files describing objects from geometry, texture, viewpoint, lighting and shading information.

MNIST. This data set is a database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples.

The scalable data sets tools are ongoing development.

According with the specification of Multimedia and data sets, we implement the W4-1 to W4-6 workloads. Details of Multimedia workloads are shown on the Table 14.

Table 14. The summary of Multimedia workloads.

ID	Implementation	Description	Data Set	Software Stack
W4-1	BasicMPEG [43]	MPEG2 decode/encode	DVD Input Streams	Libc
W4-2-1	SIFT [48]	Detect and describe local features in input images	ImageNet	MPI
W4-2-2	DBN [48]	Implementation of Deep Belief Networks	MNIST	MPI
W4-3	Speech Recognition [6]	Translate spoken words into text	English broadcasting audio files	MPI
W4-4	Ray Tracing [60]	Generating an 3D image by tracing light	Image scene	MPI
W4-5	Image Segmentation [29]	Partitioning an image into multiple segments	ImageNet	MPI
W4-6	Face Detection [61]	Detecting face in an image	ImageNet	MPI

5.5 Bioinformatics

Fig. 16. Data processing flow of genome detection.

In the Bioinformatics specification, the data flow is shown in Fig. 16. The original data set is genome sequence data, so we choose genome sequence data consisting of anopheles gambiae genome data and Ventner Homo sapiens genome data for Sequence assembly. For Sequence alignment, we choose assembly of the human genome data as the original data since these data are assembled. These data sets are described as following:

Genome sequence data[4]. This data set is unstructured, consisting of 4 genome data, with the size ranging from 20MB to 7GB, and the number of reads ranging from 101617 to 31257852.

Assembly of the human genome[5]. This data set is unstructured, including 4 assembly sequences, with the data format of fasta, and the size ranging from 100MB to 13GB.

According with the specification of Bioinformatics and data sets, we implement the W5-1 and W5-2 workloads. As shown on the Table 15

Table 15. The summary of Bioinformatics workloads.

ID	Implementation	Description	Data Set	Software Stack
W5-1	SAND	Sequence assembly implementations which merge genome fragments to get the original genome sequence	Genome sequence data	Work Queue
W5-2	BLAST	Sequence alignment implementations which identify the similarity between target sequence with sequence in database	Assembly of the human genome data	MPI

5.6 BDGS: Big Data Generation Tools

We have described the implementation of the workloads based on the real data sets. To achieve the purpose of large scale benchmarking, we should scale up these data sets. Big Data Generation tools (BDGS) is designed for scaling up the real data sets in BigDataBench. The current version of BDGS can scale up three types of data: Text, Graph and Table. The details of Generators of Text, Graph and Table are as follows.

Text Generator We implement our text generator with latent dirichlet allocation (LDA) [23] model. LDA models each document as a mixture of latent topics and a topic model is characterized by a distribution of words. The document generation process in LDA has three steps:

1. choose $N \sim \text{poisson}(\xi)$ as the length of documents.
2. choose $\theta \sim \text{Dirichlet}(\alpha)$ as the topic proportions of document.
3. for each of N words w_n:
   (a) choose a topic $z_n \sim \text{Multinomial}(\theta)$
   (b) choose a word w_n from $p(w_n|, z_n, \beta)$, a multinomial probability conditioned on the topic z_n

Figure 17 shows the process of generating text data. It first preprocesses a real data set to obtain a word dictionary. It then trains the parameters of a LDA model from this data set. The parameters α and β are estimated using a variational EM algorithm, and the implementation of this algorithm is in lda-c. Finally, we use the LDA process mentioned above to generate documents.

The **Wikipedia Entries** and **Amazon Movie Reviews** data can be scalable by Text Generator.

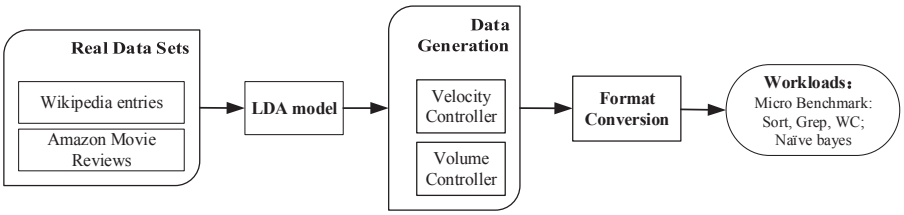


Fig. 17. The process of Text Generator in BDGS

Graph Generator We use the kronecker graph model[38, 39] in our graph generator. The kronecker graph model is designed to create self-similar graphs. It begins with an initial graph, represented by adjacency matrix N. It then progressively produces larger graphs by kronecher multiplication. Specifically, we use the algorithms in [39] to estimate initial N as the parameter for the raw real graph data and use the library in Stanford Network Analysis Platform (SNAP, http://snap.stanford.edu/) to implement our generator. Figure 18 shows the process of our Graph Generator.

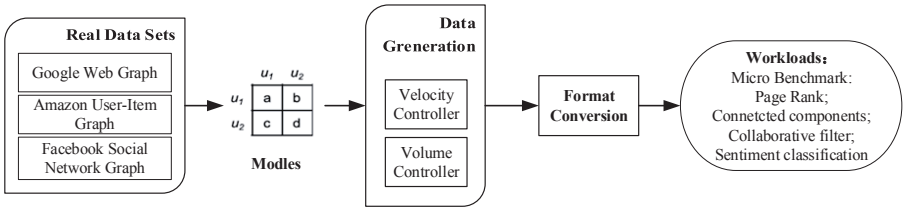


Fig. 18. The process of Graph Generator

The **Google Web Graph** and **Facebook Social Network** data can be scalable by Graph Generator.

Table Generator To scale up the E-commence transection table data, we use the PDGF [59], which is also used in BigBench and TPC-DS. PDGF uses XML configuration files for data description and distribution, thereby sampling the generation of different distributions of specified data sets. Hence we can easily use PDGF to generate synthetic table data by configuring the data schema and data distribution to adapt to real data. The **E-commerce Transaction**, **ProfSearch Person Resumés** can be scalable by Table Generator.

6 BigDataBench subsetting

6.1 Motivation

For system and architecture researches, i. e., architecture, OS, networking and storage, the number of benchmarks will be multiplied by different implementations, and hence becoming massive. For example, BigDataBench 3.1 provides about 77 workloads (with different implementations). Given the fact that it is expensive to run all the benchmarks, especially for architectural researches those usually evaluate new designs using simulators, downsizing the full range of the BigDataBench 3.1 benchmark suite to a subset of necessary (non-substitutable) workloads is essential to guarantee cost-effective benchmarking and simulations.

6.2 Methodology

1. Identify a comprehensive set of workload characteristics from a specific perspective, which affect the performance of workloads.
2. Eliminate the correlation data in those metrics and map the high dimension metrics to a low dimension.
3. Use the clustering method to classify the original workloads into several categories and choose representative workloads from each category.

The methodology details of subsetting (downsizing) workloads are summarized in [32].

6.3 Architecture subset

Microarchitectural Metric Selection We choose a broad set of metrics of different types that cover all major characteristics. We particularly focus on factors that may affect data movement or calculation. For example, a cache miss may delay data movement, and a branch misprediction flushes the pipeline.

Table 16 summarizes them, and we categorize them below.

Instruction Mix. The instruction mix can reflect a workload's logic and affect performance. Here we consider both the instruction type and the execution mode (i.e., user mode running in ring three and kernel mode running in ring zero).

Cache Behavior. The processor in our experiments has private L1 and L2 caches per core, and all cores share an L3. The L1 cache is split for instructions and data. The L2 and L3 are unified. We track the cache misses per kilo instructions and cache hits per kilo instructions except L1 data cache, note that for the L1D miss penalties may be hidden by out-of-order cores.

Translation Look-aside Buffer (TLB) Behavior. Modern processors have multi levels of TLB (most of them are two-level). The first level has separate instruction and data TLBs. The second level is shared. We collect statistics at both levels.

Branch Execution. We consider the miss prediction ratio and the ratio of branch instructions executed to those retired. These reflect how many branch instructions are predicted wrong and how many are flushed.

Pipeline Behavior. Stalls can happen in any part of the pipeline, but superscalar out-of-order processors prevent us from precisely breaking down the execution time [37, 28]. Retirement-centric analysis also has difficulty accounting for how the CPU cycles are used because the pipeline continues executing instructions even when retirement is blocked [40]. Here we focus on counting cycles stalled due to resource conflicts, e.g., reorder buffer full stalls that prevent new instructions from entering the pipeline.

Offcore Requests and Snoop Responses. Offcore requests tell us about individual core requests to the LLC (Last Level Cache). Requests can be classified into data requests, code requests, data write-back requests, and request for ownership (RFO) requests. Snoop responses give us information on the workings of the cache coherence protocol.

Parallelism. We consider Instruction Level Parallelism (ILP) and Memory Level Parallelism (MLP). ILP reflects how many instructions can be executed in one cycle (i.e., the IPC), and MLP reflects how many outstanding cache requests are being processed concurrently.

Operation Intensity. The ratio of computation to memory accesses reflects a workload's computation pattern. For instance, most big data workloads have a low ratio of floating point operations to memory accesses, whereas HPC workloads generally have high floating point operations to memory accesses ratios [62].

Removing Correlated Data The BigDataBench 3.1 includes 77 workloads. Given the 77 workloads and 45 metrics for each workload, it is difficult to analyze all the metrics to draw meaningful conclusions. Note, however, that some metrics may be correlated. For instance, long latency cache misses may cause pipeline stalls. Correlated data can skew similarity analysis — many correlated metrics will overemphasize a particular property's importance. So we eliminate correlated data before analysis. Principle Component Analysis (PCA) [36] is a common method for removing such correlated data [57, 26, 27, 22]. We first normalize metric values to a Gaussian distribution with mean equal to zero and standard deviation equal to one (to isolate the effects of the varying ranges of each dimension). Then we use Kaiser's Criterion to choose the number of principle components (PCs). That is, only the top few PCs, which have eigenvalues greater than or equal to one, are kept. With Kaiser's Criterion, the resulting data is ensured to be uncorrelated while capturing most of the original information. Finally we choose nine PCs, which retain 89.3% variance.

Clustering We use K-Means clustering on the nine principle components obtained from the PCA algorithm to group workloads into similarly behaving application clusters and then we choose a representative workload from each cluster. In order to cluster all the workloads into reasonable classes, we use the Bayesian

Table 16. Microarchitecture Level Metrics.

Category	No.	Metric Name	Description
Instruction Mix	1	LOAD	load operations' percentage
	2	STORE	store operations' percentage
	3	BRANCH	branch operations' percentage
	4	INTEGER	integer operations' percentage
	5	FP	X87 floating point operations' percentage
	6	SSE FP	SSE floating point operations' percentage
	7	KERNEL MODE	the ratio of instruction running on kernel mode
	8	USER MODE	the ratio of instruction running on user mode
	9	UOPS TO INS	the ratio of micro operation to instruction
Cache Behavior	10	L1I MISS	L1 instruction cache misses per K instructions
	11	L1I HIT	L1 instruction cache hits per K instructions
	12	L2 MISS	L2 cache misses per K instructions
	13	L2 HIT	L2 cache hits per K instructions
	14	L3 MISS	L3 cache misses per K instructions
	15	L3 HIT	L3 cache hits per K instructions
	16	LOAD HIT LFB	loads miss the L1D and hit line fill buffer per K instructions
	17	LOAD HIT L2	loads hit L2 cache per K instructions
	18	LOAD HIT SIBE	loads hit sibling core's L2 cache per K instructions
	19	LOAD HIT L3	loads hit unshared lines in L3 cache per K instructions
	20	LOAD LLC MISS	loads miss the L3 cache per K instructions
TLB Behavior	21	ITLB MISS	misses in all levels of the instruction TLB per K instructions
	22	ITLB CYCLE	the ratio of instruction TLB miss page walk cycles to total cycles
	23	DTLB MISS	misses in all levels of the data TLB per K instructions
	24	DTLB CYCLE	data TLB miss page walk cycles to total cycles
	25	DATA HIT STLB	DTLB first level misses that hit in the second level TLB per K instructions
Branch Execution	26	BR MISS	branch miss prediction ratio
	27	BR EXE TO RE	the ratio of executed branch instruction to retired branch execution
Pipeline Behavior	28	FETCH STALL	the ratio of instruction fetch stalled cycle to total cycles
	29	ILD STALL	the ratio of Instruction Length Decoder stalled cycle to total cycles
	30	DECODER STALL	the ratio of Decoder stalled cycles to total cycles
	31	RAT STALL	the ratio of Register Allocation Table stalled cycles to total cycles
	32	RESOURCE STALL	the ratio of resource related stalled to total cycles, which including load store buffer full stalls, Reservation Station full stalls, ReOrder buffer full stalls and etc
	33	UOPS EXE CYCLE	the ratio of micro operation executed cycle to total cycles
	34	UOPS STALL	the ratio of no micro operation executed cycle to total cycles
Offcore Request	35	OFFCORE DATA	percentage of offcore data request
	36	OFFCORE CODE	percentage of offcore code request
	37	OFFCORE RFO	percentage of offcore Request For Ownership
	38	OFFCORE WB	percentage of data write back to uncore
Snoop Response	39	SNOOP HIT	HIT snoop responses per K instructions
	40	SNOOP HITE	HIT Exclusive snoop responses per K instructions
	41	SNOOP HITM	HIT Modified snoop responses per K instructions
Parallelism	42	ILP	Instruction Level Parallelism
	43	MLP	Memory Level Parallelism
Operation Intensity	44	INT TO MEM	integer computation to memory access ratio
	45	FP TO MEM	floating point computation to memory access ratio

Information Criterion (BIC) to choose the proper K value. The BIC is a measure of the "goodness of fit" of a clustering for a data set. The larger the BIC scores, the higher the probability that the clustering is a good fit to the data. Here we determine the K value that yields the highest BIC score.

We use the formulation from Pelleg et al. [56] shown in Equation 1 to calculate the BIC.

$$BIC(D,K) = l(D|K) - \frac{p_j}{2} log(R) \tag{1}$$

Where D is the original data set to be clustered. In this Section, D is 77×9 matrix which indicates 77 workloads and each workload is represented by 9 PCs (Principle Components). $l(D|K)$ is the likelihood. R is the number of workloads to be clustered. p_j is the sum of $K-1$ class probabilities, which is $K+dK$. d is the dimension of each workloads, which is $K+dK$, which is 9 for we choose 9 PCs. To compute $l(D|K)$, we use Equation 2.

$$l(D|K) = \sum_{i=1}^{K} (-\frac{R_i}{2} log(2\pi) - \frac{R_i \cdot d}{2} log(\sigma^2) - \frac{R_i - K}{2} + R_i log R_i - R_i log R) \tag{2}$$

Where R_i is the number of points in the i^{th} cluster, and σ^2 is the average variance of the Euclidean distance from each point to its cluster center, which is calculate by Equation 3.

$$\sigma^2 = \frac{1}{R-K} \sum_i (x_i - \mu(i))^2 \tag{3}$$

Here x_i is the data point assigned to cluster i, and $\mu(i)$ represents the center coordinates of cluster i.

We ultimately cluster the 77 workloads (all big data workloads in BigDataBench 3.0) into 17 groups, which are listed in Table 17 .

Representative Workloads Selection There are two methods to choose the representative workload from each cluster. The first is to choose the workload that is as close as possible to the center of the cluster it belongs to. The second one is to select an extreme workload situated at the "boundary" of each cluster.

Combined with hierarchical clustering result, we select the workload situated at the "boundary" of each cluster as the architecture subset of BigDataBench 3.1. The rationale behind the approach would be that the behavior of the workloads in the middle of a cluster can be extracted from the behavior of the boundary, for example through interpolation. So the representative workloads are listed in Table 18. And the number of workloads that each selected workload represents is given in the third column.

In the case that researchers need the workloads which are chosen by the first method, i.e. choosing the workload that is as close as possible to the center of the cluster, we also list them in Table 19.

7 BigDataBench Simulator Version

We use MARSSx86 [12] and Simics [15] for our BigDataBench simulator version. This section gives a brief introduction on these two computer architecture simulators and our simulator version benchmark suite. We hope that readers can have a preliminary understanding of simulator and our BigDataBench simulator version.

7.1 Motivation

A full-system simulator is an architecture simulator that simulates an electronic system at such a level of detail that complete software stacks from real systems can run on the simulator without any modification. A full system simulator effectively provides virtual hardware that is independent of the nature of the host computer. The full-system model typically has to include processor cores, peripheral devices, memories, interconnection buses, and network connections. Architecture simulators, which aim at allowing accurate timings of the processor, are very useful in the following ways:

- Obtaining detailed performance characteristics: A single execution of simulators can generate a large set of performance data, which can be analyzed offline.
- Evaluating different hardware designs without building expensive physical hardware systems.
- Debugging on simulator to detect the potential errors instead of on real hardware, which requires re-booting and re-running the code to reproduce the problems.

We provide the BigDataBench simulator version to facilitate the big data researches in the above aspects. Simulation is a time consuming activity. It is prohibitively expensive to run all big data application in BigDataBench-v3.1. So we just deploy the architecture subset application mentioned in Section 6.3, i.e. the application in Table 18, on those two simulators and release the image as BigDataBench simulator version.

7.2 MARSSx86 Version

MARSSx86 is an open source, fast, full system simulation tool built on Qemu to support cycle-accurate simulation of superscalar homogeneous and heterogeneous multicore x86 processors [12]. MARSSx86 includes detailed models of coherent caches, interconnections, chipsets, memory and IO devices. MARSSx86 can simulate the execution of all software components in the system, including unmodified binaries of applications, operating systems and libraries.

BigDataBench MARSSx86 version overview The MARSSx86 has the following characteristics:

- Good performance and accuracy: average simulated commit rate of 200K+ instructions/second.
- Qemu based full system emulation environment with models for chipset and peripheral devices.
- Detailed models for Coherent Caches and On-Chip interconnections.

MARSSx86 user Guide
System Requirements
MARSS runs a Linux platform with the following minimum requirements:

- x86_64 CPU cores with a minimum 2GHz clock and 2GB RAM (4GB RAM is preferred).
- C/C++ compiler, gcc or icc; SCons compilation tool minimum version 1.2.
- SDL Development Libraries (required for QEMU).

Deploying MARSS and Running Big Data Applications
Once meeting the above pre-requirements, compiling MARSS is simple. What users need to do is as follows:

1. Download the appropriate MARSS installation package from the web site.
2. Extract the installation package as follows:

```
tar xf marss-0.4.tar.gz
```

3. Enter the temporary installation directory, and run the command as follows:

```
$ cd marss-0.4
$ scons -Q
```

4. By default it will compile the MARSS for single simulated core. To simulate more than one core, for SMP or CMP configuration, users should add an option 'c=NUM_CORES' to compile MARSS as shown below. This command will compile the MARSS to simulate 8 cores:

```
$ scons -Q c=8
```

5. We provide four qemu-disk-images and two qemu-network-config-scripts:
   - marss-1.img (the qemu-disk-image of master node to run Impala based workloads)
   - marss-2.img (the qemu-disk-image of slaver node to run Impala based workloads)
   - marss-3.img (the qemu-disk-image of master node to run Hadoop and Spark based workloads)
   - marss-4.img (the qemu-disk-image of slaver node to run Hadoop and Spark based workloads)
   - qemu-ifup (qemu-network-config-script for master node)
   - qemu-ifup2 (qemu-network-config-script for slaver node, you should run this script before qemu-ifup)

   To run Impala workloads of BigDataBench, you should use following commands to run MARSS:

```
master: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-1.img] -monitor stdio -net nic,macaddr=52:54:00:12:34:55 -net tap,ifname=tap1,script=[path-to-qemu-ifup2]
```

```
slaver: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-2.img] -monitor stdio -net nic -net tap,ifname=tap0,script=[path-to-qemu-ifup]
```

   To run hadoop, hive, spark, shark workloads of BigDataBench, you should use following commands to run MARSS:

```
master: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-3.img] -monitor stdio -net nic,macaddr=52:54:00:12:34:55 -net tap,ifname=tap1,script=[path-to-qemu-ifup2]
```

```
slaver: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-4.img] -monitor stdio -net nic -net tap,ifname=tap0,script=[path-to-qemu-ifup]
```

6. You can use all of the regular Qemu commands. Once the VM is booted the host's command line has became the VM console and you can start the benchmark application, issue following commands in that console:

```
(qemu) simconfig -run -stopinsns 100m -stats [stats-filename] -machine MACHINE_NAME
```

   You can find the MACHINE_NAME and hardware configuration in the marss-0.4/config path. The MACHINE_NAME should be "shared_l2" or "private_l2" if you follow the commands above.

The above paragraphs shows how to run Impala based workloads. Users can use different queries by modifying the *runMicroBenchmark.sh*. For other workloads users can boot the MARSS and use the commands in Section 9.

7.3 Simics

Simics is a full-system simulator used to run unchanged production binaries of the target hardware at high-performance speeds. It can simulate systems such as Alpha, x86-64, IA-64, ARM, MIPS (32- and 64-bit), MSP430, PowerPC (32- and 64-bit), POWER, SPARC-V8 and V9, and x86 CPUs.

BigDataBench Simics version overview We use SPARC as the instruction set architecture in our Simics version simulator benchmark suite, and deploy Solaris operation systems, for the reason that the X86 architecture are not well supported by some simulators based on Simics. For instance the Flexus [14], which is a family of component-based C++ computer architecture simulators that build on Simics Micro-Architecture Interface, do not support our-of-order mode for X86 architecture.

Simics user Guide Simics is recommended to install in the /opt/virtutech directory by using the following commands.

1. Download the appropriate Simics installation package from the website, such as simics-pkg-00-3.0.0-linux.tar.
2. Extract the installation package, the command is as follows:

```
tar xf simics-pkg-00-3.0.0-linux.tar
```

3. Enter the temporary installation directory and run the install script using the command as follows:

```
cd simics-3.0-install
sh install-simics.sh
```

4. The Simics requires a decryption key, which has been unpacked before. decode key has been cached in $HOME/.simics-tfkeys.

5. When the installation script finished, Simics has been installed in the /opt/virtutech/simics-<version>.
6. When the Simics is successfully installed, temporary installation directory can be deleted.

The detailed commands of how to run big data workloads in Simics can be found in Section 9.

8 Multi-tenancy of BigDataBench

8.1 Background of Multi-tenancy

What is Multi-tenancy Datacenters? Data center reflects the thinking that the network is the computer, which makes the amount of computing resource, storage resources and software resources linked together, then forming a huge shared virtual IT resources pool to provide services via the Internet. Data center focuses on the high concurrency, the diversity of application performance, low power, automation, high efficiency.

Within this context, a multi-tenant datacenter can be explained from three perspectives:

- Resource pooling and broad network access. Infrastructure resources such as VM, storage, and networking are pooled and shared among multiple cloud consumers.
- On-demand and elastic resource provision. Cloud consumers can get any quantity of resources at any time according to their demand
- Metered resources. Resources are charged in a Pay-as-you-go manner like electricity and water.

Existing problems Existing big data benchmarks typically focus on latency/throughput for a single run of workload performed in a dedicated set of machines. The benchmarking process is too synthetic that it does not match the typically operating conditions of real systems, where *mixes of different percentages of tenants and workloads share the same computing infrastructure*. For such an issue, benchmark suite that support real-world scenarios serving tenants with different amounts of users and heterogeneous workloads is urgently needed.

How to characterize datacenter tenants? Datacenter tenants can be characterized from three aspects:

- The number of tenants (scalability of benchmark): Does the system scale well with the number of tenants? How many tenants are able run in parallel?
- The priorities of tenants (Fairness of benchmark): How fair is the system, i.e., are the available resources equally available to all tenants? If tenants have different priorities ?
- Time line: how the number and priorities of tenants change over time?

How to characterize big data workloads? Big data workloads can be characterized from three aspects:

- Data characteristics, including data types and sources, and input/output data volumes, distributions.
- Computation semantics, including source codes (implementation logics of workloads) and the big data software stacks running the workloads.
- Job arrival patterns, including requests' arrival rate and sequences.

8.2 Definition of Multi-tenancy version

Multi-tenancy version of BigDataBench is a benchmark suite aiming to support the scenarios of multiple tenants running heterogeneous applications in cloud datacenters. Examples are latency-critical online services (e.g. web search engine) and latency-insensitive offline batch applications. The basic idea of Multi-tenancy version is to understand the behavior of realistic big data workloads (involves both service and batch application workloads) and their users. The specification of Multi-tenancy version is shown in Figure 19, this workload suite has been designed and implemented based on workload traces from real-world applications, allowing the flexible setting and replaying of these workloads according to users' varying requirements. At present, Multi-tenancy version consists of two types of representative workloads: Nutch search engine and Hadoop MapReduce workloads, which correspond to three real-world workload traces: Sougou, Facebook trace, and Google trace, respectively.

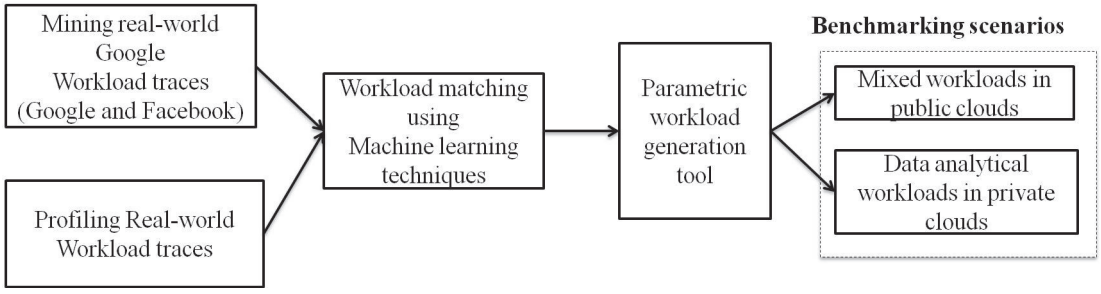

Mining real-world Google Workload traces (Google and Facebook)
Profiling Real-world Workload traces
Workload matching using Machine learning techniques
Parametric workload generation tool
Benchmarking scenarios
Mixed workloads in public clouds
Data analytical workloads in private clouds

Fig. 19. Overview of Multi-tenancy version of BigDataBench.

Main Features. Multi-tenancy version is currently integrated with Hadoop and Nutch Search Engine. We believe DC cluster operators can use Multi-

tenancy version to accomplish other previously challenging tasks, including but not limited to resource provisioning and planning in multiple dimensions; configurations tuning for diverse job types within a workload; anticipating workload consolidation behavior and quantify workload superposition in multiple dimensions.

The multi-tenancy version has the following five features:

- Repository of workload traces and real life Search-engine workloads from production systems.

- Applying robust machine learning algorithm to match the Workload characteristics information from both real workloads and workload traces, thus exacting basis for workload replaying.

- Workload synthesis tools to generate representative test workloads by parsing workload replaying basis.

- Convenient multi-tenancy workload replay tools to execute both time-critical and analytical workloads with low performance overhead.

- scenarios of both *mixed workloads in public clouds* and *data analytical workloads in private clouds.*

9 BigDataBench 3.1 user manual

9.1 BigDataBench 3.1

Table 20. The summary of the workloads in BigDataBench 3.1

Domains	Data Set	Generation Tools	Workloads	Software Stacks	ID
Search Engine	Wikipedia Entries	Text Generator	Grep	MPI, Spark, Hadoop	W1-1
			WordCount	MPI, Spark, Hadoop	W1-2
			Index	MPI, Spark, Hadoop	W1-4
			Sort	MPI, Spark, Hadoop	W1-7
	Google Web Graph	Graph Generator	PageRank	MPI, Spark, Hadoop	W1-5
	SoGou Data	N/A	Nutch Server	Nutch	W1-6
	ProfSearch Resumes	Table Generator	Read	HBase, Mysql	W1-11-1
			Write	HBase, Mysql	W1-11-2
			Scan	HBase, Mysql	W1-11-3
Social Network	Facebook Social Network	Graph Generator	CC	MPI, Spark, Hadoop	W2-8-1
			Kmeans	MPI, Spark, Hadoop	W2-8-2
	Self Generating by the program	N/A	BFS	MPI	W2-9
E-commerce	E-commerce Transaction Data	Table Generator	Select Query	Hive, Shark, Impala	W3-1
			Aggregation Query	Hive, Shark, Impala	W3-2
			Join Query	Hive, Shark, Impala	W3-3
			Project	Hive, Shark, Impala	W3-6-1
			Filter	Hive, Shark, Impala	W3-6-2
			Cross Product	Hive, Shark, Impala	W3-6-3
			OrderBy	Hive, Shark, Impala	3-6-4
			Union	Hive, Shark, Impala	W3-6-5
			Difference	Hive, Shark, Impala	W3-6-6
			Aggregation	Hive, Shark, Impala	W3-6-7
	Amazon Movie Review	Graph Generator	CF	Hadoop, Spark, MPI	W3-4
		Text Generator	Bayes	Hadoop, Spark, MPI	W3-5
Multimedia	Stream data	N/a	BasicMPEG	MPI	W4-1
	ImageNet	N/A	SIFT	MPI	W4-2-1
			Image Segmentation	MPI	W4-5
			Face Detection	MPI	W4-6
	Audio files	N/A	Speech Recognition	MPI	W4-3
	Scene description files	N/A	Ray Tracing	MPI	W4-4
	MNIST	N/A	DBN	MPI	W4-2-2
Bio-informatics	Genome sequence data	N/A	SAND	MPI	W5-1
	Assembly of the human genome	N/A	BLAST	MPI	W5-2

As shown in Table 20, we investigate five application domains including Search Engine, Social Network, E-commerce ,Multimedia and Bioinformatics, and then select representative applications/workloads from these domains. We also provide 14 real data sets, which can also be found in the table 1, for those applications. Based on the observation that the scale of real data sets may not meet the benchmarking demands of Big Data scale, we provide some data generation tool to scale out the read data.

In the table, we fill the name of corresponding data generation tool for the real data set if it can be scaled out. Users can find how to scale out the data set and run the applications in the remaining part of this section.

9.2 BigData Generation Tools

In BigDataBench 3.1,we introduce Big Data generation tools, a comprehensive suite developed to generate synthetic big data while preserving their 4V properties. Specifically, our BDGS can generate data using a sequence of three steps. First, BDGS selects application-specific and representative real-world data sets. Second, it constructs data generation models and derives their parameters and configurations from the data sets. Finally, given a big data system to be tested, BDGS generates synthetic data sets that can be used as inputs of application-specific workloads. In the release edition, BDGS consist of three parts: Text generator, Graph generator, and Table generator. We now introduce how to use these tools to generate data.

Text Generator We provide a data generation tool which can generate data with user specified data scale. In BigDataBench 3.1 we analyze the wiki data sets to generate model, and our text data generate tool can produce the big data based on the model.
Generate the data command

sh gen_text_data.sh <model_name ><file_number ><fife_lines ><line_words ><output_dir >

Parameters
<model_name >: the name of model used to generate new data
<file_number >: the number of files to generate
<fife_lines >: number of lines in each file
<line_words >: number of words in each line

For example

sh gen_text_data.sh lda_wiki1w 10 100 1000 gen_data/

This command will generate 10 files, in which each contains 100 lines, and each line contains 1000 words by using model wiki1w.
Note: The tool needs to install GSL-GNU Scientific Library. Before you run the program, Please make sure that GSL is ready.

Your also can choose parallel,

```
mkdir /mnt/raid/BigDataGeneratorSuite in every node
```

Configure NON password login and the host: parallel_ex/conf_hosts
To Run

```
cd parallel_ex
sh deploy_ex.sh
sh run_textGen_.sh
```

Graph Generator Here we use Kronecker to generate data that is both mathematically tractable and have all the structural properties from the real data set. (http://snap.st
anford.edu/snap/index.html)In BigDataBench 3.1 we analyze the Google, Facebook and Amazon data sets to generate model. Our graph data generate tool can produce the big data based on the model.
Generate the data command fill the name of corresponding data generation tool

```
sh gen_kronecker_graph
```

Parameters
-o:Output graph file name (default:'graph.txt')
-m:Matrix (in Maltab notation) (default:'0.9 0.5; 0.5 0.1')
-i:Iterations of Kronecker product (default:5)
-s:Random seed (0 - time seed) (default:0)
For example

```
sh gen_kronecker_graph -o:../data-outfile/amazon_gen.txt -m:"0.7196 0.6313; 0.4833 0.3601" -i:23
```

Note:If you want to recompilation ,you should do this

```
cd BigDataGeneratorSuite/Graph_datagen/snap-core
make
mv Snap.o ../
cd ../ and make
```

Table Generator We use Parallel Data Generation Framework (PDGF) to generate table data. PDGF is the generic data generator for database benchmarking. PDGF is designed to take advantage of today's multi-core processors and large clusters of computers to generate large amounts of synthetic benchmark data qucikly. PDGF uses a fully computational approach and it is a pure Java implementation which makes it very portable.
You can use your own configuration file to generate table data.
1. Prepare the configuration files
The configuration files are written in XML and are by default stored in the config folder. PDGF-V2 is configured with two XML files: the schema configuration and the generation configuration. The schema configuration (demo-schema.xml) defines the structure of the data and the generation rules, while the generation configuration (demo-generation.xml) defines the output and the post-processing of the generated data.
For the demo, we will generate the files demo-schema.xml and demo-generation.xml, which are also contained in the provided .gz file. Initially, we will generate two tables: OS_ORDER and OS_ORDER_ITEM.

demo-schema.xml

demo-generation.xml 2. Generate data
After creating both demo-schema.xml and demo-generation.xml a first data generation run can be performed. Therefore it is necessary to open a shell, change into the PDGF Environment directory.
Basic command-line usage WITH Scale Factor:

```
cd Table_datagen/e-com
java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf 2000
```

Your also can choose the parallel version, it runs like this

```
mkdir /mnt/raid/BigDataGeneratorSuite in every node
```

Configure Non password login and the host: parallel_ex/conf_hosts
To Run

```
cd parallel_ex
sh deploy_ex.sh
sh run_personalResumeGen.sh
```

9.3 BigData Workloads

After generating the big data, we integrate a series of workloads to process the data in our big data benchmarks. In this part, we will introduce how to run the Benchmark for each workload. It typically consists two steps. The first step is to generate the big data and the second step is to run the applications using the data we generated.

SearchEngine In SearchEngine domain,we have used data set including :Wikipedia Entries,Google Web Graph,ProfSearch Resumes and SoGou Data.The Wikipedia Entries are used by WordCount,Sort,Grep,Index workloads.The Google Web Graph is used by PageRank workload.ProfSearch Resumes are used by Cloud OLTP(Write,Read,Scan).The SoGou Data is usde by Nutch Server.
Hadoop-version (sort, grep, wordcount)
To prepare and generate data

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/MicroBenchmarks/*
3. *sh genData_MicroBenchmarks.sh*

To run

```
sh run_MicroBenchmarks.sh
```

When you choose to run sort you should do this
Put the sort-transfer file in your Hadoop_HOME(the sort-transfer you can find in BigDataBench_V3.1.tar.gz)and run like this

1. *sh genData_MicroBenchmarks.s*
2. *sh sort-transfer.sh*
3. *sh run_MicroBenchmarks.sh*

Spark-version (sort, grep, wordcount)
To prepare and generate data

1. *tar xzf BigDataBench_Sprak_V3.1.tar.gz*
2. *cd BigDataBench_V3.1_Spark_Shark/MicroBenchmarks/*
3. *sh genData_MicroBenchmarks.sh*

To run

sh run_MicroBenchmarks.sh

Mpi-version (sort, grep, wordcount)
Sort
To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1.tar.gz*
2. *cd BigDataBench_V3.1_MPI/MicroBenchmarks/MPI_Sort/*
3. *sh genData_sort.sh*

Makefile
This we provide two versions,you can choose make it by yourself, to do that you must translate like this
make
To run

mpirun -n process_number ./mpi_sort <hdfs Path ><hdfs port ><input_file ><output_file >

For example

mpirun -n 24 ./mpi_sort 172.18.11.107 9000 /home/mpi /data

Grep
To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1tar.gz*
2. *cd BigDataBench_MPI_V3.1tar.gz /MicroBenchmarks/MPI_Grep/*
3. *sh genData_grep.sh*

Then there will be a data-grep file in the current directory,you can find your data in it .If you use not one machine you must put the data on each mpi-machines,most of all you must put them in the same path.
Makefile
This we provid two version,you can choose make it by yourself ,if you do that you must translate like this
make
To run

mpirun -n process_number ./mpi_grep <input_file ><pattern >

Wordcount
To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1.tar.gz*
2. *cd BigDataBench_MPI_V3.1.tar.gz /MicroBenchmarks/MPI_WordCount/*
3. *sh genData_wordcount.sh*

Then there will be a data-wordcount file in the current directory,you can find your data in it .If you use not one machine you must put the data on each mpi-machines,most of all you must put them in the same path.
Makefile
This we provid two version,you can choose make it by yourself ,if you do that you must translate like this:
make
To run

mpirun -n process_number ./run_wordcount <input_file >

PageRank
The PageRank program now we use is obtained from Hibench.
Hadoop-version
To prepare and generate data

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/SearchEngine/PageRank*
3. *sh genData_PageRank.sh*

To run

sh run_PageRank.sh <#_Iterations_of_GenGragh >

Spark-version
To prepare and generate data

1. *tar xzf BigDataBench_Sprak_V3.1tar.gz*
2. *cd BigDataBench_Sprak_V3.1/SearchEngine/ Pagerank*
3. *sh genData_PageRank.sh*

To run

sh run_PageRank.sh <#_Iterations_of_GenGragh >

Mpi-version
To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1.tar.gz*
2. *cd BigDataBench_MPI_V3.1/SearchEngine/MPI_Pagerank*
3. *sh genData_PageRank.sh*

Makefile
This we provid two version,you can choose make it by yourself ,if you do that you must translate like this:

1.Install boost and cmake
2. *cd/BigDataBench_V3.1_MPI/SearchEngine/MPI_Pagerank/parallel-bgl-0.7.0/libs/graph_parallel/test*
3. *make distributed_page_rank_test*

To run

mpirun -n process_number ./run_PageRank <InputGraphfile ><num_ofVertex ><num_ofEdges ><iterations >

Parameters
<num_ofVertex ><num_ofEdges >these two parameters you can find in your gen_data
<num_ofEdges >: data length as L
<num_ofVertex >: 2n̂
<iterations >: n

Index
The Index program we use is obtained from Hibench.
To prepare

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/SearchEngine/Index*
3. *sh prepare.sh*

(when you do prepare.sh, you must put linux.words and words these two files in /usr/share/dict)
To run

sh run_Index.sh

Nutch Server
You can find this workload in BigDataBench_V3.1_Hadoop.tar.gz.If you want to

find data and user manual,you can get from http://prof.ict.ac.cn/DCBench/
Write Read Scan
We use YCSB to run database basic operations. We provide three ways: HBase,Cassandra and MongoDB to run operations for each operation.

To Prepare
Obtain YCSB
wget https://github.com/downloads/brianfrankcooper/YCSB/ycsb-0.1.4.tar.gz

tar BigDataBench_V3.1_Hadoop.tar.gz
cd BigDataBench_V3.0_Hadoop_Hive/BasicDatastoreOperations/ycsb-0.1.4

We name $YCSB as the path of BigDataBench_V3.0_Hadoop_Hive/BasicDatastoreOperations/ycsb-0.1.4 using the following steps.

Write
1. For HBase
Basic command-line usage

cd $YCSB
sh /bin/ycsb load hbase -P workloads/workloadc -p threads=<thread-numbers >-p columnfamily=¡family¿ -p recordcount=<recordcount-value >-p hosts=<hostip >-s >load.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<family >:In Hbase case, we used it to set database column. You should have database usertable with column family before running this command. Then all data will be loaded into database usertable with column family
<recorcount-value >:the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<hostip >:the IP of the hbase's master node.

2. For Cassandra
Before you run the benchmark, you should create the keyspace and column family in the Cassandra.
You can use the following commands to create it:

CREATE KEYSPACE usertable
with placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy'
and strategy_options = {replication_factor:2};
use usertable;
create column family data with comparator=UTF8Type and default_validation_class=UTF8Type and key_validation_class=UTF8Type;

Basic command-line usage

cd $YCSB
sh /bin/ycsb load cassandra-10 -P workloads/workloadc -p threads=<thread-numbers >-p recordcount=<recorcount-value >-p hosts=<hostips >-s >load.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<recorcount-value >: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<hostips >: the IP of cassandra's nodes. If you have more than one node you should divide with ",".

3. For MongoDB
Basic command-line usage

cd $YCSB
sh /bin/ycsb load mongodb -P workloads/workloadc -p threads=<thread-numbers >-p recordcount=<recorcount-value >-p mongodb.url=<mongodb.url >-p mongodb.databas
e=<database >-p mongodb.writeConcern=normal -s >load.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<recorcount-value >: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<mongodb.url >: this parameter should point to the mongos of the mongodb. For example "mongodb://172.16.48.206:30000".
<database >: In Mongodb case, we used it to set database column. You should

have database ycsb with collection usertable before running this command. Then all data will be loaded into database ycsb with collection usertable. To create the database and the collection, you can use the following commands:
db.runCommand({enablesharding:"ycsb"});
db.runCommand({shardcollection:"ycsb.usertable",key:{ _id:1}});

Read
1. For HBase
Basic command-line usage

cd $YCSB
sh bin/ycsb run hbase -P workloads/workloadc -p threads=<thread-numbers >-p columnfamily=<family >-p operationcount=<operationcount-value >-p hosts=<hostip >-s >tran.dat

A few notes about this command:
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<family >: In Hbase case, we use it to set database column. You should have database usertable with column family before running this command. Then all data will be loaded into database usertable with column family.
<operationcount-value >: the total operations for this benchmark. For example, when you want to load 10GB data you should set it to 10000000.
<hostip >: the IP of the hbase's master node.

2. For Cassandra
Basic command-line usage

cd $YCSB
sh bin/ycsb run cassandra -10 -P workloads/workloadc -p threads=<thread-numbers >-p operationcount=<operationcount-value >-p hosts=<hostips >-s >tran.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<operationcount-value >: the total records for this benchmark. For example, when you want to load 10GB data you should set it to 10000000.
<hostips >: the IP of cassandra's nodes. If you have more than one node you should divide with ",".

3. For MongoDB
Basic command-line usage

cd $YCSB
sh bin/ycsb run mongodb -P workloads/workloadc -p threads=<thread-numbers >-p operationcount=<operationcount-value >-p mongodb.url=<mongodb.url >-p mongodb.database=<database >-p mongodb.writeConcern=normal -p mongodb.maxconnections=<maxconnections >-s >tran.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<operationcount-value >: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<mongodb.url >: this parameter should point to the mongos of the mongodb. For example "mongodb://172.16.48.206:30000".
<database >: In Mongodb case, we used it to set database column. You should have database ycsb with collection usertable before running this command. Then all data will be loaded into database ycsb with collection usertable.
To create the database and the collection, you can use the following commands:
db.runCommand({enablesharding:"ycsb"});
db.runCommand({shardcollection:"ycsb.usertable",key:{ _id:1}});
<maxconnections >:the number of the max connections of mongodb.

Scan
1. For HBase
Basic command-line usage

cd $YCSB
sh bin/ycsb run hbase -P workloads/workloade -p threads=/textless thread-numbers >-p columnfamily=<family >-p operationcount=<operationcount-value >-p hosts=<Hostip >-p columnfamily=<family >-s >tran.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<family >: In Hbase case, we used it to set database column. You should have database usertable with column family before running this command. Then all

data will be loaded into database usertable with column family.
<operationcount-value >: the total operations for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<hostip >:the IP of the hbase's master node.

2. For Cassandra
Basic command-line usage

cd $YCSB
sh bin/ycsb run cassandra-10 -P workloads/workloade -p threads=<thread-numbers >-p operationcount=<operationcount-value >-p hosts=<hostips >-s>tran.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<operationcount-value >: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<hostips >: the IP of cassandra's nodes. If you have more than one node you should divide with ",".

3. For MongoDB
Basic command-line usage

cd $YCSB
sh bin/ycsb run mongodb -P workloads/workloade -p threads=<thread-numbers >-p operationcount=<operationcount-value >-p mongodb.url=<mongodb.url >-p mongodb.database=<database >-p mongodb.writeConcern=normal -p mongodb.maxconnections=<maxconnections >-s >tran.dat

A few notes about this command
<thread-number >: the number of client threads, this is often done to increase the amount of load offered against the database.
<operationcount-value >: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
<mongodb.url >: this parameter should point to the mongos of the mongodb. For example "mongodb://172.16.48.206:30000".
<database >: In Mongodb case, we used it to set database column. You should have database ycsb with collection usertable before running this command. Then all data will be loaded into database ycsb with collection usertable. To create

the database and the collection, you can use the following commands:
db.runCommand({enablesharding:"ycsb"});
db.runCommand({shardcollection:"ycsb.usertable",key:{ _id:1}});
<maxconnections >:the number of the max connections of mongodb.

SocialNetwork In SocialNetwork domain,we have used data set including :Facebook Social Net-work.The Facebook Social Net-work is used by CC and Kmeans workloads.The BFS workload'data is generated by itself.
K-means
The K-means program we use is obtained from Mahout.
Hadoop-version
To prepare and generate data

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/SNS/Kmeans*
3. *sh genData_Kmeans.sh*

To run

sh run_Kmeans.sh

Spark-version
To prepare and generate data

1. *tar xzf BigDataBench_Sprak_V3.1.tar.gz*
2. *cd BigDataBench_V3.1_Spark+Shark/SNS/Kmeans*
3. *sh genData_Kmeans.sh*

To run

sh run_Kmeans.sh

Mpi-version
Simple-Kmeans

To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1.tar.gz*
2. *cd BigDataBench_MPI_V3.1/SNS/Simple_Kmeans*
3. *sh Generating Image_data/color100.txt 100000 >data*

The number 100000 represent output frequency ,and the number of outbound must be more then the number of data.
Makefile
This we provide two versions,you can choose to make it by yourself, if you do that you must translate like this

mpicxx Generating.cpp -o mpi_main

To run

mpirun -np 12 ./mpi_main -i data -n 10 -o result,then you will get a new cluster file like result.

Parameters
-i:the data set of clusters
-n:the number of clusters like Kmeans'K
-o:output file
Then there will be a data in the current directory. If you use not one machine you must put the datas on each
mpi-machines, most of all you must put them in the same path.
Connected Components
The Connected Components program we used is obtained from PEGASUS.
Hadoop-version
To Prepare and generate data

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/SNS/Connected_Components*
3. *sh genData_connectedComponents.sh*

To run

sh run_connectedComponents.sh

Spark-version
To prepare and generate data:

1. *tar xzf BigDataBench_Sprak_V3.1.tar.gz*
2. *cd BigDataBench_V3.1_Spark+Shark/SNS/Connected_Components/*
3. *sh genData_connectedComponents.sh*

To run

sh run_connectComponents.sh

Mpi-version
To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1.tar.gz*
2. *cd BigDataBench_MPI_V3.1/SNS/MPI_Connect*
3. *sh genData_connectedComponents.sh*

Makefile
This we provide two versions,you can choose to make it by yourself ,if you do that you must translate like this

1.Install boost and cmake
2. *cd/BigDataBench_V3.1_MPI/SNS/Connected_Components/parallel-bgl-0.7.0/ libs/graph_parallel/test*
3. *make distributed_ramt_cc*

To run

mpirun -n process_number ./run_connectedComponents <InputGraphfile ><num_ofVertex ><num_ofEdges >

Parameters
<num_ofVertex ><num_ofEdges >:these two parameters you can find in your gen_data <num_ofEdges >: data length as L
<num_ofVertex >: 2n̂
BFS (Breath first search)
To prepare

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/MicroBenchmarks/BFS/graph500*

To run

mpirun -np PROCESS_NUM graph500/mpi/graph500_mpi_simple VERTEX_SIZE

Parameters
PROCESS_NUM: number of process;
VERTEX_SIZE: number of vertex, the total number of vertex is 2ÛVERTEX_SIZE
For example
Set the number of total running process to be 4, the vertex number to be 2Ê20, the command is:

mpirun -np 4 graph500/mpi/graph500_mpi_simple 20

E-commerce In E-commerce domain,we have used data set including :E-com merce Transaction Data and Amazon Movie Review.The Amazon Movie Review is used by CF and Bayes workloads.The E-commerce Transaction Data is used by Aggregation Query,Cross Product,Difference,Filter,OrderBy,Project,Union,Select Query,Aggregation Query and Join Query workloads.
Collaborative Filtering Recommendation
Collaborative filtering recommendation is one of the most widely used algorithm in E-commerce analysis. It aims to solve the prediction problem where the task

is to estimate the preference of a user towards an item which he/she has not yet seen. We use the RecommenderJob in Mahout(http://mahout.apache.org/) as our Recommendation workload, which is a completely distributed item-based recommender. It expects ID1, ID2, value (optional) as inputs, and outputs ID1s with associated recommended ID2s and their scores. As you know, the data set is a kind of graph data.
Before you run the RecommenderJob, you must have HADOOP and MAHOUT prepared. You can use Kronecker (see 4.2.1) to generate graph data for RecommenderJob.
To prepare and generate data

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/E-commerce*
3. *sh genData_recommendator.sh*

To run

sh run_recommendator.sh

Naive Bayes
Naive Bayes is an algorithm that can be used to classify objects into usually binary categories. It is one of the most common learning algorithms in Classification. Despite its simplicity and rather naive assumptions it has proven to work surprisingly well in practice.
We use the naivebayes in Mahout(http://mahout.apache.org/) as our Bayes workload, which is a completely distributed classifier.
When you choose to run bayes,we should use mahout-0.6.So we provide the mahout-0.6 in E-commerce. You must install and export the environment.You can do like this

1. *cd BigDataBench_V3.1/E-commerce*
2. *tar -zxvf mahout-distribution-0.6.tar.gz*
3. *export BigDataBench_V3.1/E-commerce/mahout-distribution-0.6*

and then you can run it
Hadoop-version
To prepare and generate data

1. *tar xzf BigDataBench_V3.1_Hadoop.tar.gz*
2. *cd BigDataBench_V3.1_Hadoop_Hive/E-commerce*
3. *sh genData_naivebayes.sh*

To run

sh run_naivebayes.sh

Spark-version
To prepare and generate data

1. *tar xzf BigDataBench_Sprak_V3.1.tar.gz*
2. *cd BigDataBench_V3.1_Spark+Shark/E-commerce*
3. *sh genData_naivebayes.sh*

To run

sh run_naivebayes.sh

Mpi-version
To prepare and generate data

1. *tar xzf BigDataBench_MPI_V3.1.tar.gz*
2. *cd BigDataBench_MPI_V3.1/E-commerce/MPI_naivebayes*
3. *sh genData_naivebayes.sh*

The naivebayes is special ,you should generate data in every machine.
Makefile
This we provide two version,you can choose to make it by yourself ,if you do that you must translate like this

```
mpic++ -std=c++11 -o MPI_NB MPI_NB.cpp
```

And you also can use run_sort we have already translated directly, the translated file is run_naivebayes
To run

```
mpirun -n process_number ./run_naivebayes -i <inputfile >-o <save_file >
```

Aggregation Query,Cross Product,Difference,Filter,OrderBy,Project,Union Hive Version
To prepare and generate data

```
1. cd/'$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt
2. java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf $number
```

Then you can find data in output file
Create tables and load data into tables

```
1. cd $HIVE_HOME/bin
2. sh hive
create database bigdatabench;
use bigdatabench;
create table bigdatabench_dw_order(order_id int,buyer_id int,create_date string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE;
create table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE;
load data local inpath
'$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt' overwrite into table bigdatabench_dw_order;
load data local inpath '$BigDataBench_HOME /BigDataGeneratorSuite/Table_datagen/output/OS_ORDER_ITEM.txt' overwrite into table bigdatabench_dw_item;
```

create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run

1. *cd Interactive_MicroBenchmark*
2. *sh run-MicroBenchmark.sh*

(For ease of use, we recommend that you use a local mysql server to store metadata)

Shark Version

To prepare and generate data

1. *cd / '$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt*
2. *java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf $number*

Then you can find data in output file
Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes.
Create tables and load data into tables

1. *tar zxvf MicroBenchmark.tar*
2. *cd Interactive_MicroBenchmark*
3. *shark*

create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ODER_ITEM.txt';

create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ORDER.txt';

create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run

cd Interactive_MicroBenchmark
edit free_m.sh to make sure it runs correctly.
sh runMicroBenchmark.sh

Impala Version
To prepare and generate data

1. *cd / '$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt*
2. *java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf $number*

Then you can find data in output file $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes Create tables

1. *$HIVE_HOME/bin*
2. *sh hive*

create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ODER_ITEM.txt';
create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ORDER.txt';
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run

1. *tar zxvf MicroBenchmark.tar.gz*
2. *cd MicroBenchmark*

edit free_m.sh and impala-restart.sh to make sure them run correctly.

3. *sh runMicroBenchmark.sh*

Select Query,Aggregation Query,Join Query
Hive Version

To prepare and generate data

1. *cd/'$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt*
2. *java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf $number*

Then you can find data in output file
Create tables

1. *cd $HIVE_HOME/bin*
2. *sh hive*

create database bigdatabench;
use bigdatabench;
create table bigdatabench_dw_order(order_id int,buyer_id int,create_date string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE;
create table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE;
load data local inpath ' /home/output/OS_ORDER.txt'overwrite into table bigdatabench_dw_order;
load data local inpath '$BigDataBench_V3.0_Hadoop_Hive/BigDataGeneratorSuite/Table_datagen/e-com output/OS_ORDER_ITEM.txt' overwrite into table bigdatabench_dw_item;
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run

1. *cd Interactive_Query*
2. *sh run-AnalyticsWorkload.sh*

Shark Version
To prepare and generate data

1. *cd /'$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt*
2. *java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf $number*

Then you can find data in output file
Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes.
Create tables

1. *tar zxvf InteractiveQuery.tar.gz*
2. *cd InteractiveQuery*
3. *shark*

create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ODER_ITEM.txt';

create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ORDER.txt';

create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run

1. *cd InteractiveQuery*

edit free_m.sh to make sure it runs correctly.

2. *sh runQuery.sh*

Impala Version
To prepare and generate data

1. *cd/'$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt*
2. *java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf $number*

Then you can find data in output file
Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/

to HDFS and make sure these files in different pathes.
Create tables

$HIVE_HOME/bin/hive
create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ODER_ITEM.txt';
create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '—' STORED AS TEXTFILE LOCATION 'path to OS_ORDER.txt';
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run

1. *cd InteractiveQuery*
edit free_m.sh and impala-restart.sh to make sure them run correctly.
2. *sh runQuery.sh*

Multimedia In Multimedia domain,we have used data set including :Stream data,ImageNet,Audio files,Scene description files and MNIST.The Stream data is used by BasicMPEG workload.The ImageNet is used by SIFT,ImageSegmentation and FaceDetection workloads.The Audio files are used by SpeechRecognition workload.The Scene description files is used by Ray Tracing workload.The MNIST is used DBN workloads.
MPEG-2 decode/encode
This workload is adaptations of MPEG-2 encoder/decoder <5 >, which converts video frames into a compressed bit-stream. At present, this workload is a serial version.
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/*
3. *sh getPath <data_dir ><save_file >*(using data MPEG.tar.gz)

For example

sh getPath /MPEGdec_input /Multimedia/micro/MPEG/execs MPEGdec_input

Then you will find "MPEGdec_input.path" in /Multimedia/micro/MPEG/execs
To makefile
We have provided the executable file in the directory, if you want to recompile yourself,the steps are
Makefile MPGenc

1. *cd Multimedia/micro/MPEG/MPGenc*
2. *make* (you will the mpeg2enc in execs directory)

To run

1. *cd /Multimedia/Micro/MPEG/execs*
2. *sh batch <input_path_file ><output_path_file >*

SIFT
This workload is an adaptation of David Lowe's source code <6 >, which detects and describes local features in input images. We modified it to a data parallel version using MPI.
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/*
3. *sh getPath <data_dir ><save_file >*

For example

sh getPath $ImageNet_1G/Multimedia

Then you will find "ImageNet_1G.path" in /Multimedia
Makefile
We have provided the executable file in the directory,if you want to recompile yourself, the steps are

1. *cd /Multimedia/Micro/opensift-mpi*
2. *make*

This command will create an executable file "iftfeat_mpi"under directory bin
To run

1.*cd Multimedia/Micro/opensift-mpi/bin*
2. *mpirun -n process_number -f node_file ./siftfeat_mpi input file*

Face Detection
This workload is an adaptation of flandmark source code <7 >, which detects a face in input images. We modify it to a data parallel version using MPI.
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/*
3. *sh getPath <data_dir ><save_file >*(using data ImageNet_*G.tar.gz)

For example

sh getPath $ImageNet_1G/Multimedia/Multimedia

Then you will find "ImageNet_1G.path" in/Multimedia
Makefile

1. *cd /Multimedia/App/faceDetec-mpi*
2. *cmake .*
3. *make*

Then you will find "flandmark_mpi"in file "faceDetec-mpi/cpp"
To run

1. *cd/Multimedia/App/faceDetec-mpi/cpp*
2. *mpirun -n process_number -f node_file ./flandmark_mpi <input file >*

Image Segmentation
This workload is an adaptation of Pedro Felipe Felzenszwalb's source code <8 >, which segments the input images. We modify it to a data parallel version using MPI.
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/*
3. *sh getPath <data_dir ><save_file >*(using data PPM_*G.tar.gz)

For example

sh getPath $PPM_1G/Multimedia/Multimedia

Then you will find "ImageNet_1G.path" in BigDataBench-Media
Makefile

1. *cd/Multimedia/App/segment-mpi*
2. *make*

This command will create an executable file segment_mpi under directory segment-mpi/
To run

1. *cd/Multimedia/App/segment-mpi*
2. *mpirun -n process_number -f node_file ./segment_mpi <input file >*

For more details about how to run

sh segment_mpi -h

Ray Tracing
This workload is derived from john.stone's source code <9 >,which is a parallel rendering program.
Other Prerequisite Software Packages
libjpeg : (yum install -y libjpeg-devel or source installations)
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/*
3. *sh getPath <data_dir ><save_file >*
   (using data ImageScene_*G.tar.gz
)

For example

sh getPath $ImageScene_1G/Multimedia/Multimedia/ImageScence_1G Then you will find "ImageScene_1G.path" in /Multimedia

Makefile

1. *cd /Multimedia/App/tachyon/unix*
2. *make linux-mpi*

Then you will find "linux-mpi" in /Multimedia/App/tachyon/compile
You must write your work IP in "node"
You should do this:

vim batch

Change the node_file_path and save
To run

sh batch <input file >process_number node

Speech Recognition
This workload is using CMU sphinx toolkit for speech recognition <10 >.We

write a data parallel version using MPI.
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/*
3. *sh getPath <data_dir ><save_file >*
(using data Audio_*G.tar.gz)

For example

sh getPath $Audio_1G/Multimedia/Multimedia/Audio_1G

Then you will find "Audio_1G.path" in /BigDataBench_Media
Makefile

1. *cd /Multimedia/App/speech-recg*
2. *mpic++ -o decode-mpi-cpp decode-mpi.cpp -DMODELDIR=¨pkg-config –variable=modeldir pocketsphinx‘¨pkg-config –cflags –libs pocketsphinx sphinxbase‘*

To run

mpirun -n process_number -f node_file ./decode-mpi-cpp <input file ><output file >

DBN
This project contains the MPI workloads in Deep_Learning.
Arch of DBN:
one input layer(get the input samples) + one RBM + one RBM + ... + one RBM + one output layer(for BP finetune process) = DBN Arch
Train process
Pre-training RBMs
When the input layer get sample datas, stacked RBMs will be trained one by one.
Finetune training(BP process)
After pre-training, the output layer will be trained and finetunes the whole network. That is to say, the BP process relys on stackedRBMs' pre-training.
Notice: You can run rbm.out, stackedRBMs.out and dbn.out independently. But if you want to run bp.out independently, you must run stackedRBMs.out at

first and their number of thread must be the same.
How to makefile and run workloads:
To prepare

1. *tar -zxf Multimedia.tar.gz*
2. *cd Multimedia/DBN/src*

To run
DBN
Makefile

mpic++ DBN.cpp deep.o -o DBN

To run:

mpirun -n <process >./DBN

RBM
Makefile

mpic++ RBM.cpp deep.o -o RBM

To run

mpirun -n <process >./RBM

StackedRBMS
Makefile

mpic++ StackedRBMS.cpp deep.o -o StackedRBMS

To run

mpirun -n <process >./StackedRBMS

BP
Makefile

mpic++ BP.cpp deep.o -o BP

To run

mpirun -n <process >./BP

Bioinformatics In Bioinformatics domain,we have used data set including :Genome sequence data and Assembly of the humangenome.The Genome sequence data is used by SAND workload.The Assembly of the humangenome is used by BLAST workloads.// **SAND**
You can get it from http://ccl.cse.nd.edu/software/manuals/sand.html
BLAST
You can get it from http://www.mpiblast.org/Docs/Install

9.4 BigDataBench Simulator user manual

Workloads

	Workload name
1	Hadoop-WordCount
2	Hadoop-Grep
3	Hadoop-NaiveBayes
4	Cloud-OLTP-Read
5	Hive-Differ
6	Hive-TPC-DS-query3
7	Spark-WordCount
8	Spark-Sort
9	Spark-Grep
10	Spark-Pagerank
11	Spark-Kmeans
12	Shark-Project
13	Shark-Orderby
14	Shark-TPC-DS-query8
15	Shark-TPC-DS-query10
16	Impala-Orderby
17	Impala-SelectQuery

Workloads running Users can use the following commands to drive the Simics or Marss images. We use Simics as an example, you should replace the command"./simics -c ..." with "qemu/qemu-system-x86_64 ..." mentioned above to use Marss.

Hadoop-version

Experimental environment

Cluster: one master one slaver
Software: We have already provide the following software in our images.
Hadoop version: Hadoop-1.0.2
ZooKeeper version: ZooKeeper-3.4.5
Hbase version: HBase-0.94.5
Java version: Java-1.7.0

Workloads running

Workload	Master	Slaver
Wordcount	cd /master	cd /slaver
	./simics -c Hadoopwordcount_L	./simics -c Hadoopwordcount_L
	bin/hadoop jar ${HADOOP_HOME}/hadoop-examples-*.jar wordcount /in /out/wordcount	
Grep	cd /master	cd /slaver
	./simics -c Hadoopgrep_L	./simics -c Hadoopgrep_LL
	bin/hadoop jar ${HADOOP_HOME}/hadoop-examples-*.jar grep /in /out/grep a*xyz	
NaiveBayes	cd /master	cd /slaver
	./simics -c HadoopBayes_L *	./simics -c HadoopBayes_LL
	bin/mahout testclassifier -m /model -d /testdata	
Cloud OLTP-Read	cd /master	cd /slaver
	./simics -c YCSBRead_L	./simics -c YCSBRead_LL
	./bin/ycsb run hbase -P workloads/workloadc -p operationcount=1000 -p hosts=10.10.0.13 -p columnfamily=f1 -threads 2 -s>hbase_tranunlimitedC1G.dat	

Hive-version

Experimental environment

Cluster: one master one slaver
Software: We have already provide the following software in our images.
Hadoop version: Hadoop-1.0.2
Hive version: Hive-0.9.0
Java version: Java-1.7.0

Workloads running

Workload	Master	Slaver
Hive-Differ	cd /master	cd /slaver
	./simics HiveDiffer_L	./simics -c HiveDiffer_LL
	./BigOP-e-commerce-difference.sh	
Hive-TPC-DS-query3	cd /master	cd /slaver
	./simics -c TPCDSquery3_L	./simics -c TPCDSquery3_LL
	./query3.sh	

Spark-version

Experimental environment

Cluster: one master one slaver
Software: We have already provide the following software in our images.
Hadoop version: Hadoop-1.0.2
Spark version: Spark-0.8.0
Scala version: Scala-2.9.3
Java version: Java-1.7.0

Workloads running

Workload	Master	Slaver
Spark-WordCount	cd /master	cd /slaver
	./simics -c SparkWordcount_L	./simics -c SparkWordcount_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.WordCount spark://10.10.0.13:7077 /in /tmp/wordcount	
Spark-Grep	cd /master	cd /slaver
	./simics -c Sparkgrep_L	./simics -c Sparkgrep_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.Grep spark://10.10.0.13:7077 /in lda_wiki1w /tmp/grep	
Spark-Sort	cd /master	cd /slaver
	./simics -c SparkSort_L	./simics -c SparkSort_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.Sort spark://10.10.0.13:7077 /in /tmp/sort	
Spark-Pagerank	cd /master	cd /slaver
	./simics -c SparkPagerank_L	./simics -c SparkPagerank_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.PageRank spark://10.10.0.13:7077 /Google_genGraph_5.txt 5 /tmp/PageRank	
Spark-Kmeans	cd /master	cd /slaver
	./simics -c SparkKmeans_L	./simics -c SparkKmeans_LL
	./run-bigdatabench org.apache.spark.mllib.clustering.KMeans spark://10.10.0.13:7077 /data 8 4	

Shark-version

Experimental environment

Cluster: one master one slaver
Software: We have already provide the following software in our images.
Hadoop version: Hadoop-1.0.2

Spark version: Spark-0.8.0
Scala version: Scala-2.9.3
Shark version: Shark-0.8.0
Hive version: hive-0.9.0-shark-0.8.0-bin
Java version: Java-1.7.0

Workloads running

Workload	Master	Slaver
Shark-Project Shark-Orderby	cd /master	cd /slaver
	./simics -c Sharkprojectorder_L	./simics -c Sharkprojectorder_LL
	./runMicroBenchmark.sh	
Shark-TPC-DS-query8	cd /master	cd /slaver
	./simics -c Sharkproquery8_L	./simics -c Sharkquery8_LL
	shark -f query8.sql	
Shark-TPC-DS-query10	cd /master	cd /slaver
	./simics -c Sharkproquery10_L	./simics -c Sharkquery10_LL
	shark -f query10.sql	

9.5 BigDataBench- multitenancy user manual

Environment variable configuration Configure variables at /etc/profile

HADOOP_HOME=/opt/hadoop-1.2.1
SEARCH_H0ME=/opt/search/search
cp randomwriter_conf.xsl workGenKeyValue_conf.xsl $HADOOP_HOME/conf

Prepare the input data Compile Mapreduce job WriteToHdfs.java for writing input data set

mkdir hdfsWrite
javac -classpath $HADOOP_HOME/hadoop-$HADOOP_VERSION-core.jar -d hdfsWrite WriteToHdfs..java jar -cvf WriteToHdfs.jar -C hdfsWrite/

Edit $HADOOP_HOME/conf/randomwriter_conf.xsl using configuration Parameters

Make sure the "test.randomwrite.bytes_per_map" and "java GenerateReplayScript"
files have the same [size of each input partition in bytes] parameter.

Execute the following commands

```
bin/hadoop jar WriteToHdfs.jar org.apache.hadoop.examples.WriteToHdfs -
conf
conf/randomwriter_conf.xsl workGenInput
```

Generate the replay script for FacebookTrace Use GenerateReplayScriptFB.java to create a folder that includes the script of executable
Using method

```
java GenerateReplayScriptFB.java
java GenerateReplayScript
```

[Workload file]
[Actual number of services generating clusters]
[Number of testing clusters services from user]
[Input division size (byte)]
[Input number of divisions]
[Generated replay scripts catalog]
[Inputted data directory on HDFS file system]
[Workload output mark on HDFS file system]
[Data amount of every reduce task]
[workload standard error output directory]
[Hadoop command]
[Directory of WorkGen.jar]
[Directory of workGenKeyValue_conf.xsl] **Use case**
GenerateReplayScriptFB
FB-2009_samplesKMBySort_24_times_1hr_0.tsv
600
3
67108864
10
scriptsTestFB
workGenInput
workGenOutputTest
67108864
workGenLogs
hadoop
WorkGen.jar
'/usr/lib/hadoop-1.2.1/workGenKeyValue_conf.xsl'
Prepare replay scripts for Google workload traces
When use BigDataBench-multitenancy, we need to prepare scripts to workload

replay. Here we use GenerateReplayScriptGoogle.java to generate the replay scripts
Using method

Java GenerateReplayScriptGoogle.java Java GenerateReplayScriptGoogle
[workload file directory]
[replay scripts catalog]
[shark commad]

Use Case

Java GenerateReplayScriptGoogle
job_events_part-00000-of-00500_KMnew.csv
scriptTestGoogle
shark

Preparation of using Sogou Workload Trace
Use searchTrans.py to translate sogou data log
Using method

./searchTrans.py trans logFile func1:N:M-func2

logFileosogou log file
Func1:N:M
func1, func2 are performance functions, N,M are the parameter of the function. Here we add "Segmentation" and "nodo" functions, Segmentation is used to divide log file by parameters N and M.
Use case

./searchTrans.py trans SogouQ.reduced Segmentation:24:60-nodo

Workload replay in BigDataBench- multitenancy
Execute workload replay, just execute mixWorkloadReplay.sh usig command line.
Using method
cp -r scriptsTestFB $HADOOP_HOME
cp -r scriptsTestGoogle $HADOOP_HOME

cp -f search/reqs_sogou $SEARCH_HOME/search-engine/data ./mixWorkload-Replay.sh argment

If argument is "f", only execute the Facebook trace based workload (Hadoop workloads)

If argument is "g", only execute the Google trace based workload (Shark workloads)

If argument is "s", only execute Sougou trace based workload (the Nutch search workload)

If argument is "m", execute the above three workloads in parallel

10 BigDataBench users

This section first lists the BigDataBench publications, and then summarizes the projects and research papers using or citing BigDataBench. Please note that we also list the papers using or citing DCBench or CloudRank, since we have merged these two related projects into BigDataBench as explained in 1.

10.1 BigDataBench publications

If you need a citation for BigDataBench, please cite the following papers related with your work:

1. BigDataBench: a Big Data Benchmark Suite from Internet Services. Lei Wang, Jianfeng Zhan, ChunjieLuo, Yuqing Zhu, Qiang Yang, Yongqiang He, WanlingGao, Zhen Jia, Yingjie Shi, Shujie Zhang, Cheng Zhen, Gang Lu, Kent Zhan, Xiaona Li, and BizhuQiu. The 20th IEEE International Symposium On High Performance Computer Architecture (HPCA-2014), February 15-19, 2014, Orlando, Florida, USA. [62]
2. Characterizing and Subsetting Big Data Workloads. Zhen Jia, Jianfeng Zhan, Wang Lei, Rui Han, Sally A. McKee, Qiang Yang, Chunjie Luo, and Jingwei Li. In 2014 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2014. [34]
3. Zhen Jia, Lei Wang, Jianfeng Zhan, Lixin Zhang, Chunjie Luo. Characterizing data analysis workloads in data centers. 2013 IEEE International Symposium on Workload Characterization (IISWC 2013) (Best paper award). [33]
4. CloudRank-D: Benchmarking and Ranking Private Cloud Computing System for Data Processing Applications. Chunjie Luo, Jianfeng Zhan, Zhen Jia, Lei Wang, Gang Lu, Lixin Zhang, Cheng-Zhong Xu, Ninghui Sun. Front. Comput. Sci., 2012, 6(4): 347-362. [50]
5. BDGS: A Scalable Big Data Generator Suite in Big Data Benchmarking. Zijian Ming, ChunjieLuo, WanlingGao, Rui Han, Qiang Yang, Lei Wang, and Jianfeng Zhan. Lecture note in computer sciences, extended version for the fourth workshop on big data benchmarking, 2014. [51]
6. BigOP: generating comprehensive big data workloads as a benchmarking framework. Yuqing Zhu, Jianfeng Zhan, ChuliangWeng, RaghunathNambiar,

Jingchao Zhang, Xingzhen Chen, and Lei Wang. The 19th International Conference on Database Systems for Advanced Applications (DASFAA 2014), 2014. [67]
7. BigDataBench: a Big Data Benchmark Suite from Web Search Engines. WanlingGao, Yuqing Zhu, Zhen Jia, ChunjieLuo, Lei Wang, Jianfeng Zhan, Yongqiang He, Shiming Gong, Xiaona Li, Shujie Zhang, and BizhuQiu. Third Workshop on Architectures and Systems for Big Data(ASBD 2013) in conjunction with The 40th International Symposium on Computer Architecture, May 2013. [30]
8. Xi, H., Zhan, J., Jia, Z., Hong, X., Wang, L., Zhang, L., ... Lu, G. (2011, November). Characterization of real workloads of web search engines. In Workload Characterization (IISWC), 2011 IEEE International Symposium on (pp. 15-25). IEEE. [64]

10.2 Selective research papers using BigDataBench

Cloud Data Protection Akoush et al. [19] present a system that tracks information flow using record-level lineage in Hadoop MapReduce which called MrLazy. They choose the Join workload from the BigDataBench benchmark suite as the evaluation workloads and the data set is 120GB E-commerce data.

Workload Characterization Jia et al. [34] use Principle Component Analysis (PCA) to identify the most important characteristics from 45 metrics to characterize 32 big data workloads from BigDataBench. They get seven representative big data workloads by removing redundant ones. They also find that software stacks have significant impacts on workload behaviors, even that these impacts are greater than that of the algorithms employed in user application code.

Jiang et al. [35] use hardware performance counters and a custom-made memory trace collection device to analyze the behavior of BigDataBench (Spark and Hadoop workloads), SPEC CPU2006, TPC-C, CloudSuite, and DesktopCloud workloads. They find that the behavior of the Spark in-memory computing framework differs from Hadoop or scale-out service applications, DesktopCloud and traditional high performance workloads. They also find that current Intel commodity processors are sufficiently efficient for in-memory computing.

Wei et al. [63] perform memory access pattern analysis towards both emerging big data workloads (with BigDataBench) and traditional parallel workloads. They choose five BigDataBench workloads and SPLASH-2 as the basic workloads, and find that big data workloads exhibit weak temporal and spatial locality compared to traditional workloads.

Pan et al. [54] present a study of I/O characterization of big data workloads. They choose four BigDataBench workloads as the basic workloads, and find that task slots, memory size and intermediate data compression impact on I/O Characterization of workloads deeply.

Jia et al. [32] use hardware performance counters to analyze the behavior of BigDataBench (Spark and Hadoop workloads), SPEC CPU2006, TPC-C,

CloudSuite, and DesktopCloud workloads. They find that CloudSuite do not have much difference from traditional service workloads. Data analysis workloads are different from traditional desktop, service, and HPC workloads. For BigDataBench, compared with service, analysis workload own: Large amount of application level instructions, Good locality and Low branch mis-predict ratio.

Evaluating and Optimizing Big Data Hardware Systems Quan et al. [58] evaluate State-of-art Big Data System Architectures, which included Brawny-core processors: Xeon E5310 and Xeon E5645; Wimpy-core processors: Atom D510 and TileGx36. Through the evaluations with Eight BigDataBench workloads, they make the conclusions that: there is no one-size-fits-all solution for big data, and none of the microprocessors consistently wins in terms of both performance and energy efficiency for all of our Big Data workloads. So each class of workload realizes better performance and energy efficiency on different architectures.

SSD Cache Management Liu et al. [47] implement PLC-Cache in a real-world de-duplication system. Their experimental results confirm that PLC-Cache outperforms the three classical caching algorithms (e.g., FIFO, LRU, and LFU) in terms of read latency by an average of 23.4%. They replay real-world traces collected from typical applications the hive-select, which is collected by running BigDataBench to perform a cloud database test.

Performance diagnosis and Optimization of Big Data Systems Chen et al. [24] propose an ensemble MIC-based approach to pinpoint the culprits of performance problems in the big data platform, which called InvarNet-X. They choose BigDataBench as evaluation workloads.

Evaluating and Optimizing Big Data Systems Energy Efficiency Zhou et al. [66] propose new metrics: AxPUE to measures the power usage effectiveness of IT equipment and data center systems. They choose BigDataBench as benchmarking suite.

Evaluation of Virtualization Systems Ning et al. [53] propose a new network socket library in virtualization scenario which utilizes shared memory for data transmission. They choose BigDataBench as evaluation tools.

Evaluating Programming Systems Liang et al. [44] provide a comprehensive performance evaluation of Hadoop, Spark, and DataMPI based on BigDataBench.They choose three micro benchmarks (Sort, WordCount and Grep) and two application benchmarks (K-means and Naive Bayes) as evaluation workloads.

Resource management and scheduling Liang et al. [46] propose an extended map/reduce framework called Predoop. Predoop preempts the reduce task during its idle time and allocate the released resource to the map tasks on schedule. They choose the Sort and WordCount workloads from the BigDataBench benchmark suite as the evaluation workloads.

11 Questions & Answers

This chapter lists several frequently asked questions and the corresponding answers.

Q1: I can't generate the input data of Sort workload. And the error message is: Caused by : java.io.FileNotFoundException: ToSeqFile.jar (No such file or directory)

A1: You should put the sort-transfer file (ToSeqFile.jar) into your $Hadoop_Home directory, and the sort-transfer file can be found at the BigDataBench_V3.0_Hadoop_Hive packet.

Q2: When I run Index workload, the prepare.sh cannot run correctly. The error message is: Error : number of words should be greater than 0

A2: You should make sure that these two folders (linux.words and words) are placed in the path of /usr/share/dict. And these folders can be found at the BigDataBench_V3.0_Hadoop_Hive /SearchEngine/Index directory.

Q3: Can you please elaborate on it, what is the typical ramp up period time for analytic applications for 1 GB wikipedia data.

A3: The information of the ramp-up period is as follows. If a job can finish in a short period such as 10 minutes, we just run 2 times for each benchmark. The first round is ramp-up. We collect performance data at the second round. If it needs a long time to complete, we just let the first round last several minutes, so that each node can finish several tasks, and then stop the job. We begin to collect the performance data at the second round. We do ramp-up just to warm the cache, in order to reduce the cold miss or some thing like that. For 1 GB wikipedia data, you can follow the above methods, if you like, according to how long will the job continue. Also the runtime depends on the cluster configurations and the applications you use.

Q4: When I attempt to prepare a sequence file using BigDataBench's ToSeqFile.jar, I get the following errors:

[hdfs@slavenode1 MicroBenchmarks]$ hadoop jar /usr/lib/hadoop-mapreduce/ToSeqFile.jar ToSeqFile data-MicroBenchmarks/in sort-out

Exception in thread "main" java.lang.NoClassDefFoundError: ToSeqFile$Map
at ToSeqFile.run(ToSeqFile.java:55)
at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:70)
at ToSeqFile.main(ToSeqFile.java:73)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
at org.apache.hadoop.util.RunJar.main(RunJar.java:212)
Caused by: java.lang.ClassNotFoundException: ToSeqFile$Map
at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
... 8 more

Do you have any sense of how to resolve this problem?

A4: Our Bigdatabench3.0 is based on Hadoop-1.x . As hadoop-1.x.'s API is different from that of hadoop-2.x, so the data generation tool for sort can not be used in hadoop-2.x. However hadoop-2.x provides a command to generate input data for sort, which can also be used for our sort benchmark. So you can do these as follows:
1.cd $Hadoop/share/hadoop/mapreduce
2.hadoop jar hadoop-mapreduce-examples-2.5.1.jar randomwriter -D test.randomwriter.maps_per_host=2 -D test.randomwrite.bytes_per_map=1024 /sort-data Then you can find the sort-data in HDFS.

Q5: When I run the workload, I come across the following errors:
DEBUG util.NativeCodeLoader - Trying to load the custom-built native-hadoop library...
INFO util.NativeCodeLoader - Loaded the native-hadoop library

A5: This because Hadoop use some library, called native-hadoop library, which is compiled in advance. The above error means the pre-build library do not support you architecture. So you should download the source code of Hadoop and compile the native-hadoop library manually as follows:
cd $HADOOP_HOME
ant compile-native
Copy the corresponding files in $HADOOP_HOME/build/native to your own native directory.

Q6: There is Class Not Found exception when I start Hadoop.

A6: You may use some non X86 ISA, e.g., IBM Power. The Hadoop-1.0.2 use some API that is only supported in Oracle JDK. So the some JDK like IBM JKD do not support them. So just upgrade the Hadoop version to 1.2.1. It will work.

Q7: When I ran Spark workloads, it cannot work correctly. Running command:

./run-bigdatabench cn.ac.ict.bigdatabench.Sort $MASTER /sort-out /tmp/sort

Error message is:

Exception in thread "main" java.lang.NoClassDefFoundError: scala/reflect/ClassManifest

Or

Exception in thread "main" java.lang.NullPointerException
at org.apache.spark.SparkContext$.updatedConf(SparkContext.scala:1426)
at org.apache.spark.SparkContext.¡init¿(SparkContext.scala:117)
at cn.ac.ict.bigdatabench.WordCount$.main(WordCount.scala:21)
at cn.ac.ict.bigdatabench.WordCount.main(WordCount.scala)

A7: Please do the following checking:

1) Check the version of Hadoop, and the recommended version is Hadoop-1.0.2

2) Check the version of Scala and Spark, and the recommended version is
Spark-0.8.0-incubating-bin-hadoop1
Scala-2.9.3

3) Make sure that Hadoop, Spark and Scala packets are deployed correctly.
Our experimental platform: Hadoop-1.0.2
Spark-0.8.0-incubating-bin-hadoop1
Scala-2.9.3

Q8: When I run a Spark based workload, it reports Out Of Memory problem.

A8: The Spark will store some intermediate data into memory and consuming more memory than the input data. So if the memory allocated for the each task is not enough, the OOM problem will appear. For most of the problems, users can solve by tuning the following properties according to your cluster's configuration. More information can be found at [41]

References

1. http://www.tingvoa.com.
2. ftp://ftp.tek.com/tv/test/streams/Element/index.html/.
3. http://jedi.ks.uiuc.edu/ johns/raytracer/.
4. http://ccl.cse.nd.edu/software/sand/.
5. http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/.
6. http://cmusphinx.sourceforge.net/.
7. Alexa topsites. http://www.alexa.com/topsites/global;0.
8. Amazon movie reviews. http://snap.stanford.edu/data/web-Amazon.html.
9. Daiad. http://www.daiad.eu/wp-content/uploads/2014/10/D1.2_DAIAD_Requirements_and_Architecture_v1.0
10. Facebook graph. http://snap.stanford.edu/data/egonets-Facebook.html.
11. Google web graph. http://snap.stanford.edu/data/web-Google.html.
12. Micro-architectural and system simulator for x86-based systems (MARSSx86) website. http://marss86.org/ marss86/index.php/Home.
13. mnist. http://yann.lecun.com/exdb/mnist/.

14. Simflex fast, accurate & flexible computer architecture simulation. http://parsa.epfl.ch/simflex/.
15. Simics website. http://www.windriver.com/simics/.
16. Sogou labs. http://www.sogou.com/labs/.
17. Standard performance evaluation corporation (spec) website. http://www.spec.org.
18. wikipedia. http://en.wikipedia.org.
19. S. Akoush, L. Carata, R. Sohan, and A. Hopper. Mrlazy: Lazy runtime label propagation for mapreduce. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. USENIX Association.
20. K. Ammar and M. T. Özsu. Wgb: Towards a universal graph benchmark. In *Advancing Big Data Benchmarks*, pages 58–72. Springer, 2014.
21. C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.
22. C. Bienia and K. Li. Fidelity and scaling of the PARSEC benchmark inputs. In *IEEE International Symposium on Workload Characterization*, pages 1–10, Dec. 2010.
23. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.
24. P. Chen, Y. Qi, D. Hou, and H. Sun. Invarnet-x: A comprehensive invariant based approach for performance diagnosis in big data platform. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware(BPOE)*, pages 124–140. Springer, 2014.
25. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
26. L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: Selecting representative program-input pairs. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 83–94, Sep. 2002.
27. L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism*, 5(1):1–33, 2003.
28. S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A performance counter architecture for computing accurate CPI components. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 175–184, Oct. 2006.
29. P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.
30. W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, J. Zhan, Y. He, S. Gong, X. Li, S. Zhang, and B. Qiu. Bigdatabench: a big data benchmark suite from web search engines. *The Third Workshop on Architectures and Systems for Big Data (ASBD 2013), in conjunction with ISCA 2013*, 2013.
31. Q. He, D. Jiang, Z. Liao, S. C. Hoi, K. Chang, E.-P. Lim, and H. Li. Web query recommendation via sequential query prediction. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1443–1454. IEEE, 2009.
32. Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. Characterizing data analysis workloads in data centers. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE.
33. Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. Characterizing data analysis workloads in data centers. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, pages 66–76. IEEE, 2013.

34. Z. Jia, J. Zhan, L. Wang, R. Han, S. A. McKee, Q. Yang, C. Luo, and J. Li. Characterizing and subsetting big data workloads. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE.
35. T. Jiang, Q. Zhang, R. Hou, L. Chai, S. A. Mckee, Z. Jia, and N. Sun. Understanding the behavior of in-memory computing workloads. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE.
36. I. Jolliffe. *Principal Component Analysis*. Wiley Online Library, 2005.
37. K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a Quad Pentium Pro SMP using OLTP workloads. In *International Symposium on Computer Architecture*, Jun. 1998.
38. J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Knowledge Discovery in Databases: PKDD 2005*, pages 133–145. Springer, 2005.
39. J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.
40. D. Levinthal. Cycle accounting analysis on Intel Core 2 processors. https://software.intel.com/sites/products/collateral/hpc/vtune/cycle_accounting_analysis.pdf, cited Apr. 2014.
41. D. Levinthal. Spark configuration. http://spark.apache.org/docs/0.8.0/configuration.html, cited Dec. 2014.
42. H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 107–114. ACM, 2008.
43. M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The alpbench benchmark suite for complex multimedia applications. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 34–45. IEEE, 2005.
44. F. Liang, C. Feng, X. Lu, and Z. Xu. Performance benefits of datampi: A case study with bigdatabench. *arXiv preprint arXiv:1403.3480*, 2014.
45. F. Liang, C. Feng, X. Lu, and Z. Xu. Performance characterization of hadoop and data mpi based on amdahl's second law. In *Networking, Architecture, and Storage (NAS), 2014 9th IEEE International Conference on*, pages 207–215. IEEE, 2014.
46. Y. Liang, Y. Wang, M. Fan, C. Zhang, and Y. Zhu. Predoop: Preempting reduce task for job execution accelerations. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware(BPOE-5), in conjunction with VLDB 2014*. Springer.
47. J. Liu, Y. Chai, X. Qin, and Y. Xiao. PLC-Cache: Endurable ssd cache for deduplication-based primary storage. In *In Proceeding of MSST(30th International Conference on Massive Storage Systems and Technology)*, 2014.
48. D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
49. X. Lu, M. Wasi-ur Rahman, N. S. Islam, and D. K. D. Panda. A micro-benchmark suite for evaluating hadoop rpc on high-performance networks. In *Advancing Big Data Benchmarks*, pages 32–42. Springer, 2014.
50. C. Luo, J. Zhan, Z. Jia, L. Wang, G. Lu, L. Zhang, C.-Z. Xu, and N. Sun. Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. *Frontiers of Computer Science*, 6(4):347–362, 2012.
51. Z. Ming, C. Luo, W. Gao, R. Han, Q. Yang, L. Wang, and J. Zhan. Bdgs: A scalable big data generator suite in big data benchmarking. *Proceedings of the Third Workshop on Big Data Benchmarking (WBDB2013)*, 2013.

52. E. Nakucçi, V. Theodorou, P. Jovanovic, and A. Abelló. Bijoux: Data generator for evaluating etl process quality. In *Proceedings of the 17th International Workshop on Data Warehousing and OLAP*, pages 23–32. ACM, 2014.
53. F. Ning, C. Weng, and Y. Luo. Virtualization i/o optimization based on shared memory. In *Big Data, 2013 IEEE International Conference on*, pages 70–77. IEEE, 2013.
54. F. Pan, Y. Yue, J. Xiong, and D. Hao. I/O characterization of big data workloads in data centers. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware(BPOE-4), in conjunction with ASPLOS*, 2014.
55. A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM, 2009.
56. D. Pelleg and A. W. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, pages 727–734, Jun. 2000.
57. A. Phansalkar, A. Joshi, and L. John. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *International Symposium on Computer Architecture*, Jun. 2007.
58. J. Quan, Y. Shi, M. Zhao, and W. Yang. The implications from benchmarking three big data systems. In *Big Data, 2013 IEEE International Conference on*, pages 31–38. IEEE, 2013.
59. T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch. A data generator for cloud-scale benchmarking. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, pages 41–56. Springer, 2011.
60. J. E. Stone. An efficient library for parallel ray tracing and animation. *Intel Supercomputer Users Group Conference*, 1998.
61. M. Uřičář, V. Franc, and V. Hlaváč. Detector of facial landmarks learned by the structured output SVM. In G. Csurka and J. Braz, editors, *VISAPP '12: Proceedings of the 7th International Conference on Computer Vision Theory and Applications*, volume 1, pages 547–556, Portugal, 2012. SciTePress — Science and Technology Publications.
62. L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. *The 20th IEEE International Symposium On High Performance Computer Architecture (HPCA)*, 2014.
63. W. Wei, D. Jiang, J. Xiong, and M. Chen. Exploring opportunities for non-volatile memories in big data applications. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware(BPOE-4), in conjunction with ASPLOS*, 2014.
64. H. Xi, J. Zhan, Z. Jia, X. Hong, L. Wang, L. Zhang, N. Sun, and G. Lu. Characterization of real workloads of web search engines. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 15–25. IEEE, 2011.
65. Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of the 15th international conference on World Wide Web*, pages 1039–1040. ACM, 2006.
66. R. Zhou, Y. Shi, and C. Zhu. Axpue: Application level metrics for power usage effectiveness in data centers. In *Big Data, 2013 IEEE International Conference on*, pages 110–117. IEEE, 2013.
67. Y. Zhu, J. Zhan, C. Weng, R. Nambiar, J. Zhang, X. Chen, and L. Wang. BigOP: Generating comprehensive big data workloads as a benchmarking framework. In *Database Systems for Advanced Applications*, pages 483–492. Springer, 2014.

Table 17. Clustering Results

# Cluster	Workloads
1	Cloud-OLTP-Read, Impala-JoinQuery, Shark-Difference, Hadoop-Sort, Cloud-OLTP-San, Ipala-TPC-DS-query8, Impala-Crossproduct, Impala-Project, Impala-AggregationQuery, Cloud-OLTP-Write
2	Hive-TPC-DS-query10, Hive-TPC-DS-query12-1, Hive-Difference, Hadoop-Index, Hive-TPC-DS-query6, Hive-TPC-DS-query7, Hive-TPC-DS-query9, Hive-TPC-DS-query13, Hive-TPC-DS-query12-2
3	Hive-Orderby, Hive-SelectQuery, Hive-TPC-DS-query8, Impala-SelectQuery, Hive-Crossproduct, Hive-Project, Hive-JoinQuery, Hive-AggregationQuery
4	Impala-TPC-DS-query6, Impala-TPC-DS-query12_2, Hive-TPC-DS-query3,Spark-NaiveBayes, Impala-TPC-DS-query7, Impala-TPC-DS-query13, Impala-TPC-DS-query9, Impala-TPC-DS-query10, Impala-TPC-DS-query3
5	Shark-Union, Spark-WordCount, Shark-Aggregation-AVG, Shark-Filter, Shark-Aggregation-MAX, Shark-SelectQuery, Shark-Aggregation-MIN, Shark-Aggregation-SUM
6	Impala-Filter, Impala-Aggregation-AVG, Impala-Union, Impala-Orderby, Impala-Aggregation-MAX, Impala-Aggregation-MIN, Impala-Aggregation-SUM
7	Hive-Aggregation-AVG, Hive-Aggregation-MIM, Hive-AggregationSUM, Hadoop-Grep, Hive-Union, Hive-AggregationMAX, Hive-Filter, Hadoop-Pagerank
8	Shark-TPC-DS-query9, Shark-TPC-DS-query7, Shark-TPC-DS-query10, Shark-TPC-DS-query3
9	Shark-AggregationQuery, Shark-TPC-DS-query6, Shark-Project, Shark-TPC-DS-query13
10	Shark-JoinQuery, Shark-Orderby, Shark-Crossproduct
11	Spark-Kmeans
12	Shark-TPCDS-query8
13	Spark-Pagerank
14	Spark-Grep
15	Hadoop-WordCount
16	Hadoop-NaiveBayes
17	Spark-Sort

Table 18. Treat the marginal ones as representative workloads

No.	Workload name	Number of workloads in its cluster
1	Cloud-OLTP-Read	10
2	Hive-Difference	9
3	Impala-SelectQuery	9
4	Hive-TPC-DS-query3	9
5	Spark-WordCount	8
6	Impala-Orderby	7
7	Hadoop-Grep	7
8	Shark-TPC-DS-query10	4
9	Shark-Project	4
10	Shark-Orderby	3
11	Spark-Kmeans	1
12	Shark-TPC-DS-query8	1
13	Spark-Pagerank	1
14	Spark-Grep	1
15	Hadoop-WordCount	1
16	Hadoop-NaiveBayes	1
17	Spark-Sort	1

Table 19. Treat the central ones as representative workloads

No.	Workload name	Number of workloads in its cluster
1	Cloud-OLTP-Write	10
2	Hive-TPC-DS-query13	9
3	Hive-AggregationQuery	9
4	Impala-TPC-DS-query6	9
5	Shark-Union	8
6	Impala-Aggregation-MAX	7
7	Hive-Aggregation-AVG	7
8	Shark-TPC-DS-query7	4
9	Shark-TPC-DS-query6	4
10	Shark-Crossproduct	3
11	Spark-Kmeans	1
12	Shark-TPC-DS-query8	1
13	Spark-Pagerank	1
14	Spark-Grep	1
15	Hadoop-WordCount	1
16	Hadoop-NaiveBayes	1
17	Spark-Sort	1

Table 21. Spark Properties

Property Name	Default	Meaning
spark.executor.memory	512m	Amount of memory to use per executor process, in the same format as JVM memory strings (e.g. 512m, 2g).
spark.shuffle.consolidateFiles	false	If set to "true", consolidates intermediate files created during a shuffle. Creating fewer files can improve filesystem performance for shuffles with large numbers of reduce tasks. It is recommended to set this to "true" when using ext4 or xfs filesystems. On ext3, this option might degrade performance on machines with many (>8) cores due to filesystem limitations.
spark.shuffle.file.buffer.kb	100	Size of the in-memory buffer for each shuffle file output stream, in kilobytes. These buffers reduce the number of disk seeks and system calls made in creating intermediate shuffle files.
spark.default.parallelism	8	Default number of tasks to use for distributed shuffle operations (groupByKey, reduceByKey, etc) when not set by user.
spark.cores.max	(infinite)	When running on a standalone deploy cluster or a Mesos cluster in "coarse-grained" sharing mode, how many CPU cores to request at most. The default will use all available cores offered by the cluster manager.